www.keithley.com

Series 3700A System Switch/Multimeter

Reference Manual

3700AS-901-01 Rev. B / May 2013

3700AS-901-01

A Greater Mesure of Confidence

KEITHLEY

A Tektronix Company

Series 3700A

System Switch/Multimeter

Reference Manual

Document number: 3700AS-901-01 Rev. B / May 2013

Safety precautions

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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The symbol indicates a connection terminal to the equipment frame.

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

Safety precaution revision as of January 2013.

Table of Contents

Section 1

Introduction

In this section:

Welcome

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

Extended warranty

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

Contact information

If you have any questions after you review the information in this documentation, please contact your local Keithley Instruments representative or call Keithley Instruments corporate headquarters (toll-free inside the U.S. and Canada only) at 1-888-KEITHLEY (1-888-534-8453), or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (*http://www.keithley.com*).

CD-ROM contents

Two CD-ROMs are shipped with each Series 3700A order. The Series 3700A Quick Start Guide, User's Manual, Reference Manual, and Switch Card Manual are provided in PDF format on the Series 3700A Product Information CD-ROM.

- **Quick Start Guide:** Provides unpacking instructions, describes basic connections, and reviews basic operation information. If you are new to Keithley Instruments equipment, refer to the Quick Start Guide to take the steps needed to unpack, set up, and verify operation.
- **User's Manual:** Provides application examples. If you need a starting point to begin creation of applications, refer to the User's Manual for a variety of specific examples.
- **Reference Manual:** Includes advanced operation topics and maintenance information. Programmers looking for a command reference, and users looking for an in-depth description of the way the instrument works (including troubleshooting and optimization), should refer to the Reference Manual.
- **Switching and Control Cards Reference Manual:** Contains information specific to the switch cards that can be used with the Series 3700A.
- **Additional product information:** The product data sheet, product specifications, and rack-mount kit instructions are also included on the CD-ROM.

A second CD-ROM contains the Test Script Builder script development software (Keithley Instruments part number KTS-850). Use this CD-ROM to install the Test Script Builder Integrated Development Environment. This software provides an environment to develop a test program and the ability to load the test program onto the instrument. Running a program loaded on the instrument eliminates the need to send individual commands from the host computer to the instrument when running a test.

The second CD-ROM also includes:

- The 3700A TSB Add-in, which is a software tool you can use to create, modify, debug, and store Test Script Processor (TSP®) test scripts.
- IVI Instrument Driver, driver for National Instruments LabVIEW™, and related release notes.
- J2SE™ Runtime Environment: Web browser plug-in required to run the web applications that are available through the instrument web interface.
- Keithley I/O layer and release notes.
- Keithley LXI Discovery Browser.

For the latest drivers and additional support information, see http://www.keithley.com (*http://www.keithley.com*).

Organization of manual sections

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

- **General overview:** Describes the components of the instrument and basic operation.
- **Basic DMM operation:** Introduces some basic measurement functions of the instrument. You will learn how to use your instrument to measure voltage, current, resistance, frequency, period, continuity, diodes, capacitance, and temperature. You will learn about triggering and data buffering. You will also learn about enhancing measurement performance.
- **Theory of operation:** Describes basic DMM measurement techniques and concepts.
- **Remote commands:** Describes the IEEE Std 488.2 common commands. Provides an alphabetical listing of all commands available for the Series 3700A. Provides information about using remote commands to control the instrument.
- **Troubleshooting guide:** Describes self-test procedure and error codes.
- **Maintenance:** Information on instrument maintenance, including line fuse replacement and firmware upgrades.
- **Status model:** Description of the Series 3700A status model.

Bookmarks for each section of this manual are provided in the PDF version of the documentation. The manual sections are also listed in the Table of Contents located at the beginning of this manual.

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

Capabilities and features

The Series 3700A System Switch/Multimeter is comprised of four versions of the Model 3706A system switch mainframe and a series of plug-in switch and control cards.

Series 3700A available models

Model number	Description
3706A	Six-slot system switch with high-performance digital multimeter (DMM)
3706A-NFP	Six-slot system switch with high-performance digital multimeter (DMM) without front-panel display and keypad
3706A-S	Six-slot system switch
3706A-NFP	Six-slot system switch without front-panel display and keypad

Available plug-in cards

Model number	Description
3720	Dual 1x30 multiplexer card (auto CJC when used with optional Model 3720-ST screw-terminal accessory)
3721	Dual 1x20 multiplexer card (auto CJC when used with optional Model 3721-ST screw-terminal accessory)
3722	Dual 1x48 high-density multiplexer card
3723	Dual 1x30 high-speed reed relay multiplexer card
3724	Dual 1x30 FET multiplexer card
3730	6x16 high-density matrix card
3731	6x16 high-speed reed relay matrix card
3732	Quad 4x28 ultra-high density reed relay matrix card
3740	32-channel isolated switch card
3750	Multifunction control card

Series 3700A instruments have the following features:

- Six slot system switch with optional high-performance multimeter
- Three remote interfaces: LXI/ethernet, general purpose bus (GPIB), and Universal Serial Bus (USB)
- Fourteen programmable digital I/O lines
- Up to 576 2-wire or 720 1-wire multiplexer channels in one mainframe
- Up to 2,688 one-pole matrix crosspoints in one mainframe
- Capable of more than 14,000 readings per second to memory with high-performance multimeter option
- Filtering to reduce reading noise
- Internal memory stores multiple user setups
- USB flash drive access for saving data buffers, test scripts, and user setups
- Digital I/O port: Allows the Series 3700A to control other devices
- LXI® Class C compliance
- Embedded TSP scripting engine accessible from any host interface; responds to high-speed test scripts comprised of instrument control commands
- TSP-Link® expansion bus that allows TSP-enabled instruments to trigger and communicate with each other; advanced Test Script Processor (TSP®) scripting engine features enable parallel script execution across the TSP-Link network
- Supports IEEE-488 (GPIB), RS-232, and ethernet local area network (LAN) connections

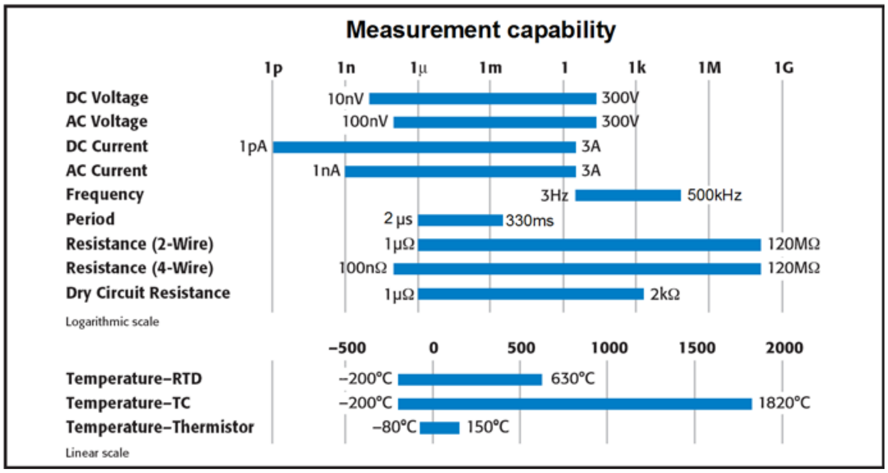
Measuring capabilities

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

Reviewers, this figure is not correct for the Series 3700A. However, it shows the type of information that will be in the final manual.

Mary, add capatance.

Figure 1: DMM measurement capabilities



General information

Displaying the instrument's serial number

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel using the front-panel keys and menus.

To display the serial number on the front panel:

1. If the Series 3700A is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Press the **MENU** key.
3. Use the navigation wheel to scroll to the **UNIT-INFOSYSTEM-INFO** menu item.
4. Press the **ENTER** key. The SYSTEM INFORMATION menu is displayed.
5. Scroll to the **SERIAL#** menu item.
6. Press the **ENTER** key. The Series 3700A serial number is displayed.

General operation

In this section:

Turning your instrument on and off

The following topics describe how to power your instrument on and off, place an instrument in standby, and configure the line frequency.

Procedure

The Series 3700A operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. At the factory, each Series 3700A is configured to match the power line frequency appropriate for your country (either 50 Hz or 60 Hz). Make sure the operating voltage in your area is compatible.

Follow the procedure below to connect the Series 3700A to line power and turn on the instrument.

CAUTION

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

To turn a Series 3700A on and off:

1. Before plugging in the power cord, make sure that the front panel POWER switch is in the off (O) position.
2. Connect the Model 3706A redundant protective earth (safety ground) located on the Rear panel.
3. Connect the female end of the supplied power cord to the AC receptacle on the rear panel.

WARNING

The power cord supplied with the Series 3700A contains a separate protective earth (safety ground) wire for use with grounded outlets. When proper connections are made, the instrument chassis is connected to power-line ground through the ground wire in the power cord. In addition, a redundant protective earth connection is provided through a screw on the rear panel. In the event of a failure, not using a properly grounded protective earth or grounded outlet may result in personal injury or death due to electric shock.

4. Connect the other end of the power cord to a grounded AC outlet.
5. To turn your instrument on, press the front panel **POWER** switch to place it in the on (I) position.
6. To turn your instrument off, press the front panel **POWER** switch to place it in the off (O) position.

Line frequency configuration

The factory configures the Series 3700A to automatically detect the power line frequency (either 50 Hz or 60 Hz) at each power-up. This detected line frequency is used for aperture (NPLC) calculations.

To view the line frequency setting, send the following command:

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

Fuse replacement

A rear panel fuse drawer is located below the AC receptacle (refer to Rear panel). This fuse protects the power line input of the instrument. If the line fuse needs to be replaced, refer to Line fuse replacement (on page A-1).

Power-up sequence

When the instrument is turned on, the instrument performs self-tests on its read-only memory, nonvolatile memory, and RAM on its read-only memory, nonvolatile memory, and RAM and momentarily lights all segments and indicators. If a failure is detected, the instrument momentarily displays an error message and the ERR indicator turns on. Error messages are listed in Error and status messages (on page 9-10, on page 2-60, "*Error and status message list*" on page 9-12).

If there are no errors, the following actions occur when the instrument is turned on:

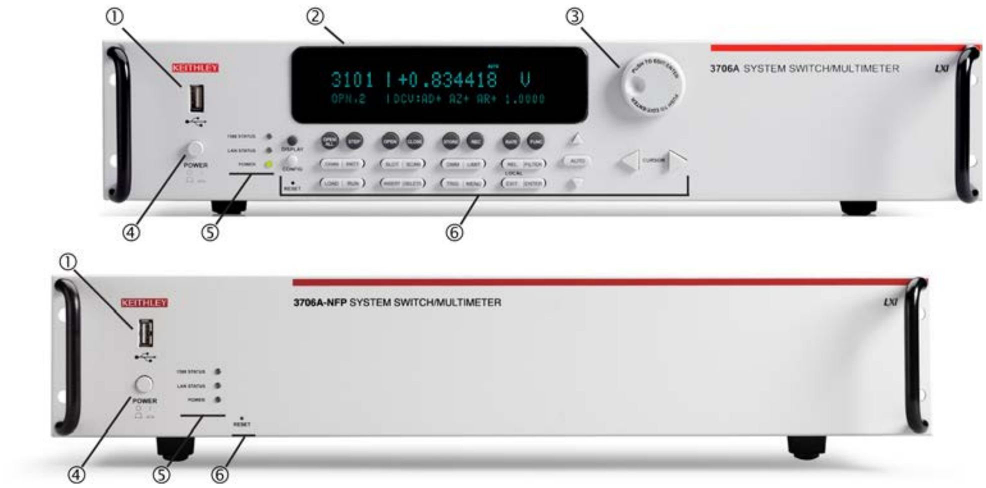
1. "No Comm Link" is briefly displayed.
2. "Initializing" is displayed for several seconds.
3. Near the end of initialization, the 1588 and LAN status LEDs light.
4. All of the display pixels briefly light.
5. Main display is displayed.

Front panel operation

The Series 3700A includes several models that support different features. The following figures show the front panels of each of the models; a brief description of the features follows the figures.

Figure 2: Model 3706A with DMM front panel

Figure 3: Model 3706A-S front panel (no DMM)



(1) The USB port

(USB port)

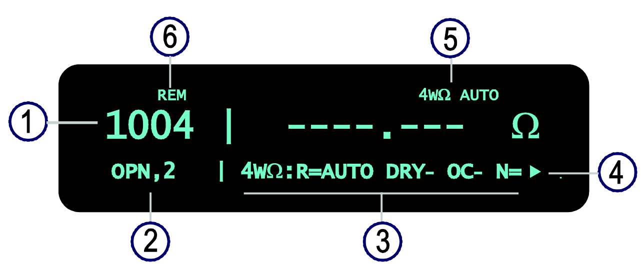
Use the front-panel USB port to connect a USB flash drive. The USB flash drive can be used to store reading buffer data, scripts, and user setup options.

(2) The display

During setup, the display shows menu choices that you can use to configure the instrument. See Menu overview (on page 2-9) for more information about Series 3700A menus.

During operation, the display provides information about the selected channel, channel pattern, channel state, and errors. It also shows the control status (local or remote). If `REM` is displayed, the instrument is presently controlled through a remote interface (GPIB, LAN, or USB). If `REM` is not displayed, control is through the front panel. The following figure shows an example of the Series 3700A during operation.

Figure 4: Series 3700A display during operation



Series 3700A display during operation	
1	Active channel (slot 1, channel 004).
2	Channel state (open, 2-pole operation).
3	Present state of the DMM attributes for displayed channel: • The 4-WΩ and autorange are enabled • Dry-circuit ohms is disabled (DRY-) • Offset compensation is off (OC-) For detailed descriptions of the DMM attribute symbols, see the table labeled "DMM attribute symbols" below.
4	Arrow indicating that more menu items exist; turn the navigation wheel to the left or right to see the additional items.
5	The 4-WΩ and autorange are enabled.
6	Indicates the instrument is being controlled remotely. Press the LOCAL (EXIT) key to control the instrument through the front panel.

The table below lists the display indicators and what they mean.

Indicator	Meaning
AUTO:	Measure autorange is selected
EDIT:	Instrument is in the editing mode
ERR:	Questionable reading or invalid calibration step
FILT:	Digital filter is enabled
LSTN:	Instrument is addressed to listen
MATH:	Enabled for mX+b, percent, or reciprocal (1/X) calculation
REL:	Relative mode is enabled
REM:	Instrument is in remote mode
SRQ:	Service request is asserted
TALK:	Instrument is addressed to talk
TRIG:	Instrument is processing a front-panel reading request
4W:	Four-wire resistance or RTD temperature reading
***** (asterisk):	Readings are being stored in the buffer

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

DMM attribute symbols

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

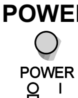
(3) The navigation wheel

Turn the navigation wheel to scroll to the desired menu option or to change the value of the selected numeric parameter. Pressing the navigation wheel has the same functionality as pressing the **ENTER** key.

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

(4) The POWER key

POWER



Power switch. The in position turns the Series 3700A on (I); the out position turns it off (O).

(5) The status lights

The Series 3700A has three status lights on the front panel.

The 1588 status light indicates 1588 operation. If this light is off, the 1588 feature is disabled or improperly configured. If the light blinks at a one second rate, the instrument is the 1588 master. If the instrument is a slave, the light will not blink.

The LAN status light is lit when the instrument is connected through the local area network (LAN) with no errors. If this is not lit, the instrument is not connected through the LAN or there is a connection problem.

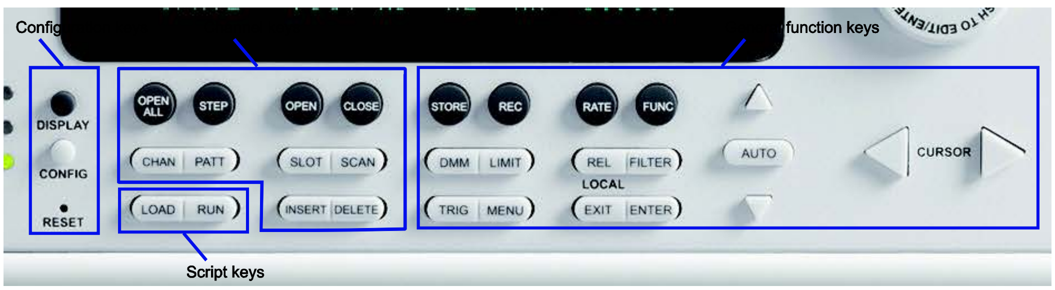
If you are using the web interface, the LAN Status light blinks when you click **ID**.

The POWER light is lit when power is applied to the instrument.

(6) The setup and control keys

The setup and control keys provide front-panel control and configuration. The following figure illustrates each key's location. Descriptions of the keys follow the figure.

Figure 5: Model 3706A setup and control keys



Configuration keys

DISPLAY key	The **DISPLAY** key cycles between three screens: The channel display or pattern display, the closed channel list, and the user screen text, which is set with display.settext() (on page 8-146). When the closed channel listing is displayed, if the list of channels is longer than one screen, you can use the navigation wheel to scroll though the list of closed channels.
CONFIG key	The **CONFIG** key configures a function or operation.
RESET switch	The **RESET** switch restores factory default LAN settings.

Channel keys

OPEN ALL key	Opens all closed channels.
STEP key	Use to walk though a scan list by closing and opening the channels contained in a single step with each press of the key.
CLOSE and **OPEN** keys	You can use the front-panel CLOSE and OPEN keys to perform either switch only operations or switch with DMM operations on the selected channels. The operations of the keys depend on the DMM configuration attribute setting of the selected channel. Refer to Channel attributes (on page 2-93) for more information on the DMM configuration attribute. When the DMM configuration is set to `nofunction`, the CLOSE and OPEN keys function as switch only operations in the same manner as the `channel.close` and `channel.open` commands. When the DMM configuration is associated with a particular function (for example, DC Volts), the CLOSE and OPEN keys function as switch with DMM operations, that is, in the same manner as `dmm.close` and `dmm.open` commands. To access the other switch-only operations (exclusive close and exclusive slot close), use the CHAN key to choose and initiate the desired operation after selecting a channel or range of channels.
CHAN key	If a channel is displayed, opens the CHANNEL ACTION menu options (on page 2-14), which allows you to open and close channels. If a pattern is displayed, pressing CHAN switches to channel view.
PATT key	If a pattern is displayed, opens the PATTERN ACTION menu options (on page 2-15), which allows you to manage patterns, open and close patterns, and reset them. If a channel is displayed, pressing PATT changes to display a pattern.
SLOT key	Displays information about the installed cards and the instrument. Information includes the firmware revision, model name, and model number. Press SLOT multiple times to view all instrument information.
SCAN key	Opens the SCAN ACTION menu options (on page 2-16), which allows you to run, manage, view, and reset scan lists. See Scanning and triggering (on page 3-1).
INSERT key	Appends the selected channel or channel pattern to the scan list.
DELETE key	Deletes the first occurrence of the selected channel or channel pattern (including function) from the scan list. To remove all occurrences of a channel from the list, keep pressing the DELETE key.

Script keys

LOAD key	Loads test for execution.
RUN key	Runs the last selected user-defined test code.

General function keys

STORE key	Selects, clears, and saves reading buffer data and creates and deletes reading buffers.
REC key	Recalls stored readings for the selected reading buffer. Use the **CURSOR** keys or turn the navigation wheel to scroll through the buffer. For more information, see Recalling readings (on page 3-52).
RATE key	Sets measurement speed (fast, medium, or slow) for the active or selected function.
FUNC key	Displays a menu that allows you to scroll through the available DMM functions.
DMM key	Opens the DMM ACTION menu options (on page 2-16).
LIMIT key	Set the limits. Press multiple times to cycle through the four combinations of limit settings: • Limit1 and Limit2 off • Limit1 on and Limit2 off • Limit1 off and Limit2 on • Limit1 and Limit2 on
REL key	Enable or disable relative offset for the selected DMM function. REL is shown on the display when relative offset is enabled. See Relative.
FILTER key	Enables or disables the digital filter; you can use this filter to reduce reading noise.
TRIG key	Generates a trigger that can be used in a script or the trigger model. See Scanning and triggering (on page 3-1). Also see display.trigger.EVENT_ID (on page 8-147).
MENU key	Opens the Main menu options, which allows you to manage scripts, manage communications, select channel connections, test the keys, test the display, manage digital I/O settings, set up the beeper, and display instrument information.
EXIT (LOCAL) key	• Cancels the current selection and returns to the previous menu item. • Exits remote operation so you can control the instrument from the front panel. • Stops a scan that is running. • Stops a script that is executing.
ENTER key	Accepts the current selection or brings up the next menu option. In most cases, pressing ENTER is the same as pressing the navigation wheel.
AUTO key	Enables or disables autorange for the selected function.
RANGE keys (up and down arrows)	Selects the next higher or lower measurement range on the measurement display for the selected function. If the Model 3706A displays the overflow message on a particular range, select a higher range until an on-range reading is displayed. Use the lowest range possible without causing an overflow to ensure best accuracy and resolution. You can also use these keys when entering a range value from the front panel. For details, see Autoranging over the front panel (see "Set up autoranging from the front panel" on page 4-50). In addition to selecting range functions, the up and down range keys change the format for non-range numbers (as an example, when editing the limit value). If you select a range of channels, that range must stop when the channel type changes. Therefore, you can never select a range of channels which includes different channel types. For more information, see Range.
CURSOR keys	Use the CURSOR keys to move the cursor left or right. When the cursor is on the desired compliance value digit, push the navigation wheel to enter edit mode, and turn the navigation wheel to edit the value. Push the navigation wheel again when finished editing. Use the CURSOR keys or the navigation wheel to move through menu items. To view a menu value, use the CURSOR keys for cursor control, and then press the navigation wheel to view the value or sub-menu item.

Menu overview

Menu navigation

To navigate through the menus and submenus, the Series 3700A must not be in edit mode (the EDIT indicator is not illuminated).

Selecting menu items

To navigate the Main and Configuration menus, use the editing keys as follows:

- Press either **CURSOR** arrow key to highlight an option.
- Rotate the navigation wheel (clockwise or counter-clockwise) to highlight an option.
- Press the **ENTER** key (or the navigation wheel) to select an option.
- Use the **EXIT (LOCAL)** key to cancel changes or to return to the previous menu or display.

Quick Tip

For quick menu navigation, turn the navigation wheel to highlight an option and then press the navigation wheel to select the highlighted option.

Setting a value

You can adjust values on the front panel using the navigation wheel:

1. Use the **CURSOR** arrow keys to move the cursor to the value that you want to edit.
2. Press the navigation wheel or the **ENTER** key to enter edit mode. The EDIT indicator is illuminated.
3. Rotate the navigation wheel to set the appropriate value.
4. Press the **ENTER** key to select the value or press the **EXIT (LOCAL)** key to cancel the change.
5. (Optional) Press the **EXIT (LOCAL)** key to return to the main menu.

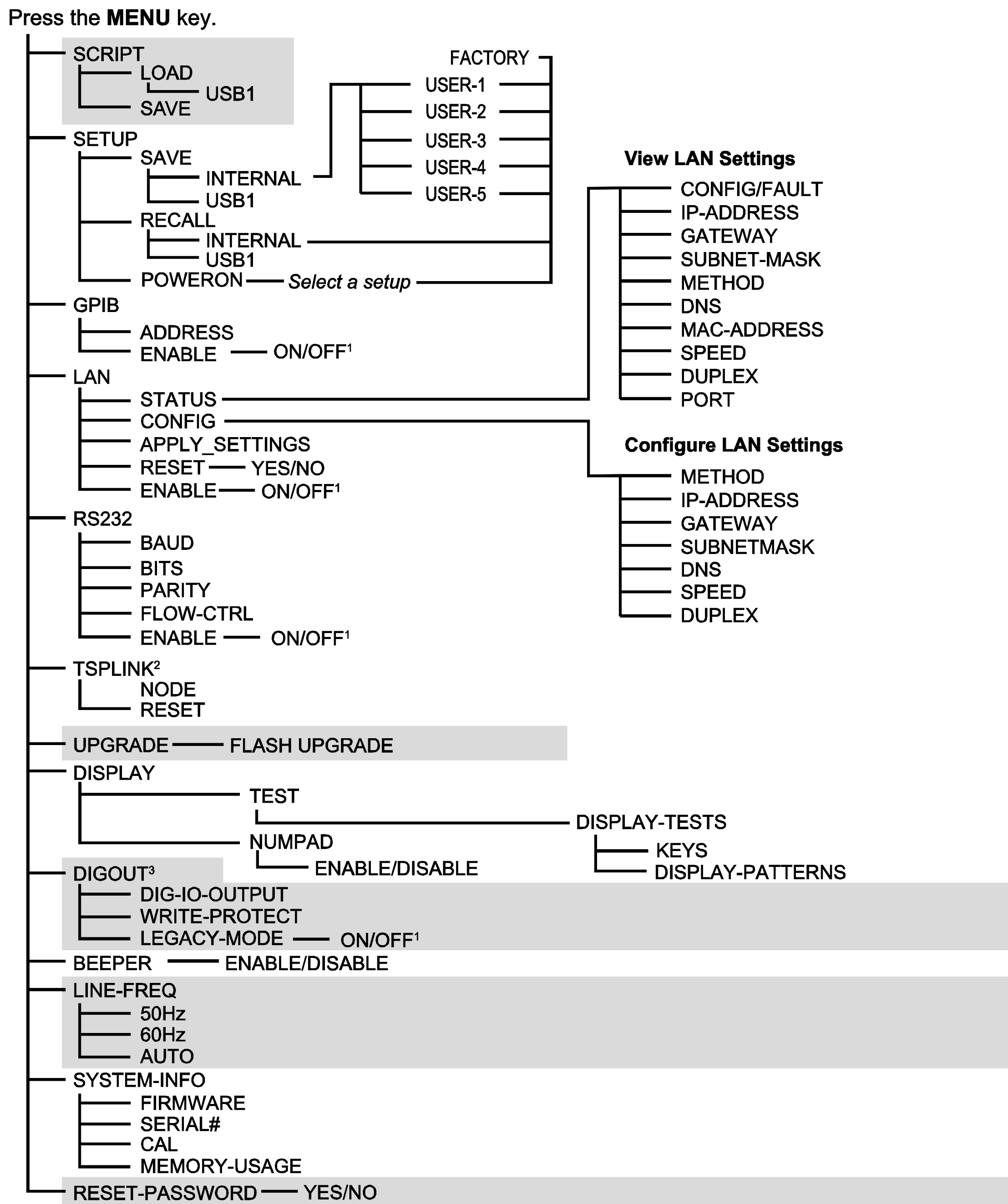
Menu trees

You can configure instrument operation through the menus that are accessed from the front panel.

Main menu

The main menu structure is summarized in the following figure and table. For other menu items, see Configuration menus (on page 2-12).

Figure 6: Main menu tree



1. Mutually exclusive
2. TSPLINK is not available on the Models 2604B, 2614B, and 2634B.
3. DIGOUT is not available on the Models 2604B, 2614B, and 2634B.

The following table contains descriptions of the main menu options and cross-references to related information. To access a menu option, press the **MENU** key, turn the navigation wheel to move the cursor to select an item, and press the navigation wheel.

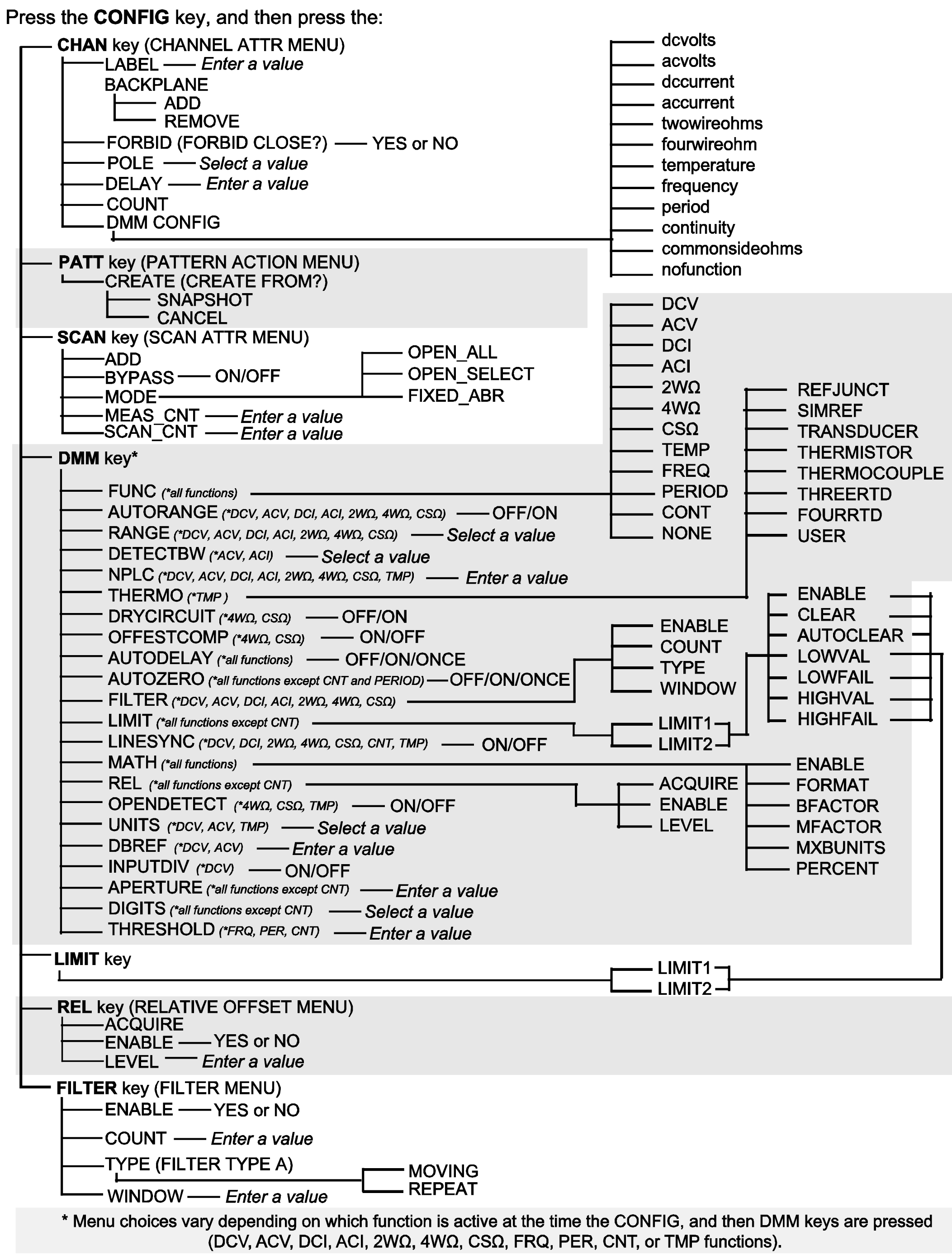
Menu selection	Description	For more information, see:
SCRIPT - LOAD - SAVE	Saves and recalls users scripts Loads scripts into nonvolatile memory Saves scripts	Manage scripts (on page 7-3)
SETUP - SAVE - RECALL - POWERON	Saves and recalls user and factory setup options Saves user setup options Recalls user setup options Sets the configuration used during startup	Saved setups (on page 2-33)
GPIB - ADDRESS - ENABLE	Configures the GPIB interface options Configures the address for the GPIB interface Enables and disables the GPIB interface	Remote communication interfaces (on page 2-53)
LAN - STATUS - CONFIG - APPLY_SETTINGS - RESET - ENABLE	Configures the local area network (LAN) Displays LAN connection status Configures the LAN IP address and gateway Applies changes made using the CONFIG menu Restores the default settings Enables and disables the LAN interface	LAN concepts and settings (on page B-1)
RS232 - BAUD - BITS - PARITY - FLOW-CTRL - ENABLE	Controls the options for the RS-232 interface Sets the baud rate Configures the number of bits Sets the parity Configures the flow control Enables and disables the RS-232 interface	Remote communication interfaces (on page 2-53)
TSPLINK[1] - NODE - RESET	Configure the instrument in a TSP-Link® network Selects the instrument node identifier Resets the TSP-Link network	TSP-Link system expansion interface (on page 7-45)
UPGRADE	Upgrades the firmware from a USB flash drive	Upgrading the firmware (on page A-6)
DISPLAY - TEST - NUMPAD	Accesses display functions Runs the display test Enables and disables the numeric keypad	Front panel tests See Numeric entry method in Setting a value (on page 2-9)
DIGOUT[2] - DIG-IO-OUTPUT - WRITE-PROTECT	Controls digital outputs Selects the digital I/O values Write-protects specific digital I/O lines	Digital I/O (on page 3-43)
BEEPER - ENABLE - DISABLE	Controls the key beeps Enables the key beeps Disables the key beeps	General operation (on page 2-1)
LINE-FREQ - 50Hz - 60Hz - AUTO	Configures the line frequency Set the line frequency to 50 Hz Set the line frequency to 60 Hz Enables automatic line frequency detection during start up	General operation (on page 2-1)
SYSTEM-INFO - FIRMWARE - SERIAL# - CAL - MEMORY-USAGE	Displays the system information Displays the version of firmware installed Displays the serial number of the unit Displays the last calibration date Displays memory usage in percentage	General operation (on page 2-1)
RESET-PASSWORD	Resets the system password	Password management

1. TSPLINK is not available on the Models 2604A, 2614A, and 2634A.
2. DIGOUT is not available on the Models 2604A, 2614A, and 2634A.

Configuration menus

The configuration menu structure is summarized in the following figure and table. For directions on navigating the menu, see Menu navigation (on page 2-9). For other menu items, see Main menu (on page 2-10).

Figure 7: Series 3700A configuration menus



Quick Tip

Press the **EXIT** key to return to a previous menu.

MEAS	V and I-measure range, V-measure sense, low range; autozero	Range, Basic source-measure procedure
LIMIT	V-source and I-source compliance limits	Limits
SPEED	Measurement speed (NPLC)	Speed
REL	Set relative values	Relative offset (on page 4-41)
FILTER	Control digital filter	Filters
OUTPUT ON/OFF	Set off-state, control digital I/O	Output-off states
TRIG	Set trigger in, count, interval, and delay	Triggering
STORE	Set buffer count and destination	Source-measure concepts

The following table contains descriptions of the configuration menus, as well as cross-references to related information. To select a menu, press the **CONFIG** key and then the front-panel key associated with the menu (see the description column in the following table).

Configuration menu options

Configuration menu	**To access, press the** CONFIG **key and then:**	**Description**	**For more information, see:**
CHANNEL ATTRIBUTE	**CHAN**	If a channel is displayed when selecting this, configure channels; if a channel pattern is displayed when you select this, change channels states in the pattern	CHAN key configuration (on page 2-17)
PATTERN ACTION	**PATT**	Manage, open and close, and reset patterns	PATT key configuration (see "SCAN key configuration" on page 2-20)
SCAN ATTR	**SCAN**	Run, manage, view, and reset scan lists	SCAN key configuration (on page 2-20)
DMM	**DMM**	Manage measurement settings like measurement speed (NPLC)	Speed DMM key configuration (see "LIMIT key configuration" on page 2-23)
LIMIT	**LIMIT**	Manage limit 1 and 2 settings for the active function	DMM measurement capabilities (on page 4-1) LIMIT key configuration (on page 2-23)
RELATIVE OFFSET	**REL**	Set relative values	Relative offset (on page 4-41) REL key configuration (on page 2-23)
FILTER	**FILTER**	Manage the digital filter settings	Filters FILTER key configuration (on page 2-23)
FUNCTION	**FUNC**	Set DMM functions	FUNC key configuration (on page 2-24)
RD BUFFER ATTR	**STORE**	If a buffer has been selected when you press the key, you can view and set the reading buffer attributes	STORE key configuration (on page 2-24)

Front-panel key menu options

The menus that can be accessed from the front panel of the instrument allow you to set up and run the instrument.

LOAD TEST menu options

Allows you to run scripts and code from the front panel that you created through the communication interface, or configuration scripts created by pressing the front-panel MENU key, then selecting SCRIPT > CREATE-CONFIG.

To open this menu, press **LOAD**.

The User option loads code that was added to Load Test with the display.loadmenu.add() (on page 8-138) command.

The Scripts option loads named scripts that were added to the run-time environment. See Manage scripts (on page 7-3) for information on creating and loading scripts.

After selecting code or script from the User or Scripts option, you can press **RUN** to execute the selected code or script.

CHANNEL ACTION menu options

Allows you to change the state of channels from the front panel.

To open this menu, display a channel, then press **CHAN**.

Switch channel options include:

- **OPEN**: Opens the selected channel.
- **CLOSE**: Closes the selected channel.
- **EXCLOSE**: Closes the selected channel and opens any closed channels on the instrument.
- **EXSLOTCLOSE**: Closes the specified channel and opens any closed channels on the same slot. Channels on other slots remain closed.
- **RESET**: Restores the factory default settings to the selected channel. Resetting a channel deletes any channel patterns that contain that channel.

DIGIO channel options include:

- **READ**: Displays a value from a channel as 8-bit binary. This menu option does not appear if a range of channels is selected. Related command: channel.read() (on page 8-86).
- **WRITE**: Writes a value to a channel. Enter the value as 8-bit binary. Related command: channel.write() (on page 8-109).
- **RESET-STATE**: Resets the channel state. Related command: channel.resetstatelatch() (on page 8-89).
- **RESET**: Restores the factory default settings of selected channels or all channels. Related command: channel.reset() (on page 8-87).

TOTALIZER channel options include:

- **READ**: Displays a value from a channel as a number between 0 and 65535. This menu option does not appear if a range of channels is selected. Related command: channel.read() (on page 8-86).
- **WRITE**: Writes a value to a channel. Enter the value between 0 and 65535. Related command: channel.write() (on page 8-109).
- **RESET-STATE**: Resets the channel state. Related command: channel.resetstatelatch() (on page 8-89).
- **RESET**: Restores the factory default settings of selected channels or all channels. Related command: channel.reset() (on page 8-87).

DAC channel options include:

- **READ**: Displays a value from a channel. This menu option does not appear if a range of channels is selected. A number is displayed that is dependent on the channel's selected mode function, as well as the card model of the selected channel. Related command: channel.read() (on page 8-86).
- **WRITE**: Writes a value from a channel. This menu option does not appear if a range of channels is selected. A number is displayed that is dependent on the channel's selected mode function, as well as the card model of the selected channel. Related command: channel.write() (on page 8-109)
- **RESET-STATE**: Resets the channel state. Related command: channel.resetstatelatch() (on page 8-89).
- **RESET**: Restores the factory default settings of selected channels or all channels. Related command: channel.reset() (on page 8-87).

For more information, see Working with channels (on page 2-89).

PATTERN ACTION menu options

Allows you to configure and change patterns from the front panel.

To open this menu, in pattern view, press **PATT**.

Options include:

- **CREATE**: If no patterns have been created, this is the only option that is displayed. Allows you to create a new pattern.
- **OPEN**: Opens the channels in the selected channel pattern.
- **CLOSE**: Closes the channels in the selected channel pattern. These closures are appended to any channels that are already closed.
- **EXCLOSE**: Closes the channels in the selected pattern so that the channels associated with the pattern are exclusively closed. Any previously closed channels are opened.
- **EXSLOTCLOSE**: Exclusively closes the channels in the specified channel pattern for the selected slots.
- **VIEW**: Displays the channels that are in the selected pattern.
- **DELETE**: Deletes the channel pattern.
- **RESET**: Displays options that allow you to reset the channels in the selected channel pattern to factory default settings. Resetting a channel pattern causes that pattern to be deleted because when channels are reset, they delete patterns that contain them.

For information about working with channel patterns, see Channel patterns (on page 2-96).

SCAN ACTION menu options

Allows you to work with the scan lists from the front panel. You must have a scan list created before using this option. See Basic scan procedure (on page 3-5) for information.

To open this menu, press **SCAN**.

Options include:

- **BACKGROUND**: Runs the scan while allowing front panel operation.
- **CREATE**: Reminder that you must use the INSERT key to create a scan list.
- **LIST**: Displays the scan list. Use the navigation wheel ◷ to scroll through the channels.
- **CLEAR**: Clears the scan list.
- **RESET**: Resets the scan settings to the factory default settings, which includes clearing the scan list.

DMM ACTION menu options

Press the DMM key to open the DMM ACTION menu.

The DMM ACTION menu contains the following items:

- **MEASURE:** Takes measurements on the digital multimeter (DMM) without using the trigger model. Related command: dmm.measure() (on page 8-213).
- **COUNT:** Indicates the number of measurements to take when a measurement is requested. Related command: dmm.measurecount (on page 8-214).
- **LOAD:** Recalls a user or factory DMM configuration. Use the navigation wheel to scroll through available configurations. Related command: dmm.configure.recall() (on page 8-173).

- **SAVE:** Creates a DMM configuration with the pertinent attributes based on the selected function, and associates it with the specified name. Related command: dmm.configure.set() (on page 8-175).
- **OPEN:** Opens the specified channel and/or channel pattern. Related command: dmm.open() (on page 8-219).
- **CLOSE:** Closes the specified channel or channel pattern in preparation for a DMM measurement. Related command: dmm.close() (on page 8-167).
- **RESETFUNC:** Returns the DMM aspects of the system for only the active function to factory default settings. Related command: dmm.reset() (on page 8-228).
- **RESETALL:** Returns all DMM functions of the instrument to the factory default settings. Related command: dmm.reset() (on page 8-228).

Configuration menu options

CHAN key configuration

The CHAN key configuration menus will display different submenus depending on the type of channel you are using (SWITCH, DIGIO, TOTALIZER, or DAC). The following topics describe the CHAN key configuration menus by channel type.

CONFIG CHAN key - SWITCH channel type

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

When changing attribute settings for a range of channels, the menu option for the first channel specified in the range is highlighted. For example, selecting channels 3 to 5 on slot 3 on the front panel (3003:3005) as a range shows the current attribute setting for 3003 when an attribute menu is displayed.

When the attribute setting is selected for a range, the entire range of channels is updated to that value. To view or set an individual attribute setting for only one channel, be sure to select a single channel range. For example, 3003:3003 would only affect channel 3 on slot 3, which is displayed as 3003 with the channel state and poles setting below it.

The CHAN ATTR menu contains:

LABEL: Sets the label associated with the specified channel. From the front panel, the label can be up to 12 characters. Remotely, the label may be up to 19 characters. This option will not be displayed if multiple channels are selected. Related command: channel.setlabel().

BACKPLANE: Opens the BACKPLANE menu. Use this menu to add or remove backplane channels from the specified channels. Related command: channel.setbackplane().

FORBID: Allows or prevents the closing of the specified channels. Related commands: channel.setforbidden() and channel.clearforbidden().

POLE: Sets the number of poles for the specified channels. Related command: channel.setpole().

DELAY: Sets additional delay time for the specified channels. Related command: channel.setdelay().

COUNT: Displays closure cycles for the specified channel. This option is not displayed if multiple channels are selected. Related command: channel.getcount().

DMM-CONFIG: Sets the DMM configuration associated with the specified channels. Related command: dmm.setconfig().

CONFIG CHAN key - DIGIO channel type

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

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

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

LABEL: Enter up to 12 characters for the label for a channel. Related command: channel.setlabel() (on page 8-94).

DELAY: Enter the value for the delay in 1ms steps from 0 to 60 seconds for a channel. Related command: channel.setdelay() (on page 8-93).

MODE: Sets the mode attribute on a channel. Select INPUT, OUTPUT, or OUTPUT_PROTECTED. Related command: channel.setmode() (on page 8-98).

MATCH: Sets the match value on a channel. Enter the value as 8-bit binary. Related command: channel.setmatch() (on page 8-96).

MATCH-TYPE: Sets the match type on a channel. Select EXACT, ANY, NOT_EXACT, or NONE. Related command: channel.setmatchtype() (on page 8-97).

STATE: Queries for the state of a channel and displays the value in the top line, labeled by `STATE=`. Related command: channel.getstate() (on page 8-75).

CONFIG CHAN key - TOTALIZER channel type

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

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

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

LABEL: Enter up to 12 characters for the label for a channel. Related command: channel.setlabel() (on page 8-94).

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

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

Related command: channel.setmode() (on page 8-98).

MATCH: Sets the match value on a channel. Enter a value between 0 and 65535. Related command: channel.setmatch() (on page 8-96).

- **MATCH TYPE**: Sets the match type on a channel. Select EXACT, ANY, NOT_EXACT, or NONE. Related command: channel.setmatchtype() (on page 8-97).
- **STATE**: Queries for the state of a channel and displays the value in the top line, labeled by STATE=. Related command: channel.getstate() (on page 8-75).
- **POWER**: Sets the power state attribute on a channel. Select ENABLE or DISABLE. Related command: channel.setpowerstate() (on page 8-103)

CONFIG CHAN key - DAC channel type

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

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

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

NOTE

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

LABEL: Enter up to 12 characters for the label for a channel. Related command: channel.setlabel() (on page 8-94).

DELAY: Enter the value for the delay in 1 ms steps from 0 to 60 seconds for a channel. Related command: channel.setdelay() (on page 8-93).

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

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

Related command: channel.setmode() (on page 8-98).

- **OUTPUT**: Sets the output enable attribute on a channel. Select ENABLE or DISABLE. Related command: channel.setoutputenable() (on page 8-100).
- **STATE**: Queries for the state of a channel and displays the value in the top line, labeled by STATE=. Related command: channel.getstate() (on page 8-75).
- **POWER**: Sets the power state attribute on a channel. Select ENABLE or DISABLE. Related command: channel.setpowerstate() (on page 8-103).

PATT key configuration

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

The PATTERN ATTR menu contains the following item:

- **DMM_CONFIG**: Sets the DMM configuration associated with the specified channel pattern. Use the navigation wheel to scroll through the available DMM configurations. Related command: dmm.setconfig() (on page 8-237).

SCAN key configuration

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

The SCAN ATTR menu contains the following items:

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

NOTE

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

- **BYPASS:** Enables or disables bypassing the first item in the scan. Related command: scan.bypass (on page 8-324).
- **MODE:** Sets the `scan.mode` value to one of the following:
  - OPEN_ALL, which is equivalent to scan.MODE_OPEN_ALL or 0 (default setting)
  - OPEN_SELECT, which is equivalent to scan.MODE_OPEN_SELECTIVE or 1
  - FIXED_ABR, which is equivalent to scan.MODE_FIXED_ABR or 2

  Related command: scan.mode() (see "scan.mode" on page 8-331)
- **MEAS_CNT:** Sets the measure count value. Related command: scan.measurecount (on page 8-330)
- **SCAN_CNT:** Sets the scan count value. Related command: scan.scancount (on page 8-335)

DMM key configuration

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

Each function only has access to the applicable attributes for that function. Brief definitions of the available attributes are contained in the following paragraphs. Refer to the appropriate command for additional attribute information in the TSP commands (on page 8-10).

The DMM ATTR menu contains:

APERTURE: Configures the aperture setting for the active DMM function in seconds. Related command: dmm.aperture (on page 8-150).

AUTODELAY: Configures the auto delay setting for the active DMM function. Related command: dmm.autodelay (on page 8-154).

AUTORANGE: Configures the auto range setting for the DMM. Related command: dmm.autorange (on page 8-155).

AUTOZERO: Configures the autozero setting for the DMM, which periodically measures internal voltages to help maintain the stability and accuracy of the instrument over time and changes in temperature. Related command: dmm.autozero (on page 8-157). Also see Autozero.

DBREF: Configures the DB reference setting for the DMM in volts. Related command: dmm.dbreference (on page 8-178).

DETECTBW: Configures the detector bandwidth setting for the selected DMM function. For more information, see Bandwidth (on page 4-53). Related command: dmm.detectorbandwidth (on page 8-179).

DIGITS: Configures the display digits setting for the selected DMM function. For more information, see Digits programming (see "Change the display resolution" on page 4-6). Related command: dmm.displaydigits (on page 8-180).

DRYCIRCUIT: Configures the dry circuit setting for the selected DMM function. Related command: dmm.drycircuit (on page 8-181).

FILTER: Opens the FILTER menu for the selected DMM function. See FILTER key configuration (on page 2-23).

FUNC: Displays a menu that allows you to scroll through the available DMM functions. Use the navigation wheel or **CURSOR** keys to scroll the menu options and press **ENTER** when the desired function is highlighted. Related command: dmm.func (on page 8-187).

INPUTDIV: Enables or disables the 10M Ω input divider. Related command: dmm.inputdivider (on page 8-190).

LIMIT: Opens the LIMIT menu for the selected DMM function. See LIMIT key configuration (on page 2-23).

LINESYNC: Enables or disables line sync during measurements. Related command: dmm.linesync (on page 8-203).

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

- **ENABLE:** Enables or disables math operation on measurements. Related command: dmm.math.enable (on page 8-206).
- **FORMAT:** Specifies the math operation to perform on measurements. Related command: dmm.math.format (on page 8-208).
- **BFACTOR:** Specifies the offset for the y = mX + b operation. Related command: dmm.math.mxb.bfactor (on page 8-209).
- **MFACTOR:** Specifies the scale factor for the y = mX + b operation. Related command: dmm.math.mxb.mfactor (on page 8-210).
- **MXBUNITS:** Specifies the unit character for the y = mX + b operation. Related command: dmm.math.mxb.units (on page 8-211).
- **PERCENT:** Specifies the constant to use for the percent operation. Related command: dmm.math.percent (on page 8-212).

For more information, see:

- mX+b (on page 4-44)
- Reciprocal (1/X) (on page 4-47)
- Percent (on page 4-45)

NPLC: Configures the integration rate in line cycles for the DMM. Related command: dmm.nplc (on page 8-217).

OFFSETCOMP: Configures the offset compensation setting for the DMM. Related command: dmm.offsetcompensation (on page 8-218).

OPENDETECT: Configures the state of the thermocouple or 4-wire ohms open detector that is being used. Related command: dmm.opendetector (on page 8-221).

RANGE: Configures the range of DMM for the selected function for one channel type. For more information, see Range. Related command: dmm.range (on page 8-222).

REL: Opens the relative offset menu for the selected DMM function. See REL key configuration (on page 2-23).

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

- **REFJUNCT:** Allows selection of the reference junction to use. Available choices are: SIMULATED, EXTERNAL, or INTERNAL. Related command: dmm.refjunction (on page 8-223).
- **SIMREF:** Specifies the simulated reference temperature for thermocouples. Related command: dmm.simreftemperature (on page 8-239).
- **TRANSDUCER:** Selects the transducer type (THERMOCOUPLE, THERMISTOR, 3RTD, or 4RTD). Related command: dmm.transducer (on page 8-244).
- **THERMISTOR:** Specifies the type of thermistor. Related command: dmm.thermistor (on page 8-240).
- **THERMOCOUPLE:** Specifies the thermocouple type. Related command: dmm.thermocouple (on page 8-241).
- **THREERTD:** Specifies the type of 3-wire RTD. Related command: dmm.threertd (on page 8-242).
- **FOURRTD:** Specifies the type of 4-wire RTD. Related command: dmm.fourrtd (on page 8-186).
- **USER:** Specifies USER type of RTD (ALPHA, BETA, DELTA, or ZERO). Related commands: dmm.rtdalpha (on page 8-229), dmm.rtdbeta (on page 8-231), dmm.rtddelta (on page 8-233), dmm.rtdzero (on page 8-234).

THRESHOLD: Configures the threshold range. Related command: dmm.threshold (on page 8-243).

UNITS: Configures the units for voltage and temperature measurements. Related command: dmm.units (on page 8-245).

LIMIT key configuration

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

These menus contain the following items:

- **ENABLE:** Enables or disables limit testing. Related command: dmm.limit[Y].enable (on page 8-193).
- **CLEAR:** Clears the test results of the limit. Related command: dmm.limit[Y].clear() (on page 8-192).
- **AUTOCLEAR:** Indicates if the limit should be cleared automatically or not. Related command: dmm.limit[Y].autoclear (on page 8-191).
- **LOWVAL:** Sets the low limit value. Related command: dmm.limit[Y].low.value (on page 8-201).
- **LOWFAIL:** Queries for the low test results of the limit. Related command: dmm.limit[Y].low.fail (on page 8-199).
- **HIGHVAL:** Sets the high limit value. Related command: dmm.limit[Y].high.value (on page 8-197).
- **HIGHFAIL:** Queries for the high test results of limit. Related command: dmm.limit[Y].high.fail (on page 8-195).

REL key configuration

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

The RELATIVE OFFSET menu contains the following menu items:

- **ACQUIRE:** Acquires an internal measurement to store as the REL level value. Related command: dmm.rel.acquire() (on page 8-224).
- **ENABLE:** Enables or disables relative measurement control for the DMM. Related command: dmm.rel.enable (on page 8-225).
- **LEVEL:** Sets a specific offset value to use for relative measurements for the DMM. Related command: dmm.rel.level (on page 8-226).

FILTER key configuration

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

The FILTER menu contains the following menu items:

- **ENABLE**: Enables or disables filtered measurements for the selected DMM function. Related command: dmm.filter.enable (on page 8-183).
- **COUNT**: Indicates the filter count setting for the selected DMM function. Related command: dmm.filter.count (on page 8-182).
- **TYPE**: Indicates the filter averaging type for the DMM measurements on the selected DMM functions (MOVING or REPEAT). Related command: dmm.filter.type (on page 8-184).
- **WINDOW**: Indicates the filter window for the DMM measurements (0 to 10% in 0.1% increments). Related command: dmm.filter.window (on page 8-185).

FUNC key configuration

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

STORE key configuration

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

This menu contains the following menu items:

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

Using the front panel with non-switch channels

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

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

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

The following table lists the front panel channel attributes that indicate the various channel mode settings (remote command equivalent channel.setmode() (on page 8-98)), channel output enable (remote command equivalent channel.setoutputenable() (on page 8-100)), and channel label (remote command equivalent channel.setlabel() (on page 8-94)). Some of the attributes have alternate symbols, depending on the operation you are performing on the front panel and whether it is being used with the 6 or 12 character label symbol.

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

Front-panel channel attributes for channel settings

Front-panel channel setting	Symbol	Definition	Symbol meaning
Channel label	`XXXXXX`	First 6 characters of label	Used with single letter symbols
	`XXXXXXXXXXXX`	First 12 characters of label	Used with the non-single letter symbols
Digital I/O mode settings	`DIG IN`	Digital input mode	Used with 12-character label or no label
	`DIG OUT`	Digital output mode	Used with 12-character label or no label
	`DIG pOUT`	Digital output protected mode	Used with 12 character label or no label
	`I (uppercase "i")`	Digital input mode	Used with 6-character label
	`O`	Digital output mode	Used with 6-character label
	`P`	Digital output protected mode	Used with 6-character label
Totalizer mode settings	`Rise Ed`	Totalizer rising edge mode	Used with 12-character label or no label
	`Fall Ed`	Totalizer falling edge mode	Used with 12-character label or no label
	`Rise-TTL`	Totalizer rising edge TTL level mode	Used with 12-character label or no label
	`Fall-TTL`	Totalizer falling edge TTL level mode	Used with 12-character label or no label
	`Rise-RST`	Totalizer rising edge read reset mode	Used with 12-character label or no label
	`Fall-RST`	Totalizer falling edge read reset mode	Used with 12-character label or no label
	`RiseTRST`	Totalizer rising edge TTL read reset mode	Used with 12-character label or no label
	`FallTRST`	Totalizer falling edge TTL read reset mode	Used with 12-character label or no label
	`R`	Totalizer rising edge mode	Used with 6-character label
	`F`	Totalizer falling edge mode	Used with 6-character label
DAC mode settings	`V`	Voltage function mode	Used with 6-character label
	`I (uppercase "i")`	Current function either 1 or 2 mode	Used with 6-character label
	`V1`	Voltage function 1 mode	Used with 12-character label or no label
	`I1`	Current function 1 mode	Used with 12-character label or no label
	`I2`	Current function 2 mode	Used with 12-character label or no label
	`pV1`	Protected voltage function 1 mode	Used with 12-character label or no label
	`pI1`	Protected current function 1 mode	Used with 12-character label or no label
	`pI2`	Protected current function 2 mode	Used with 12-character label or no label

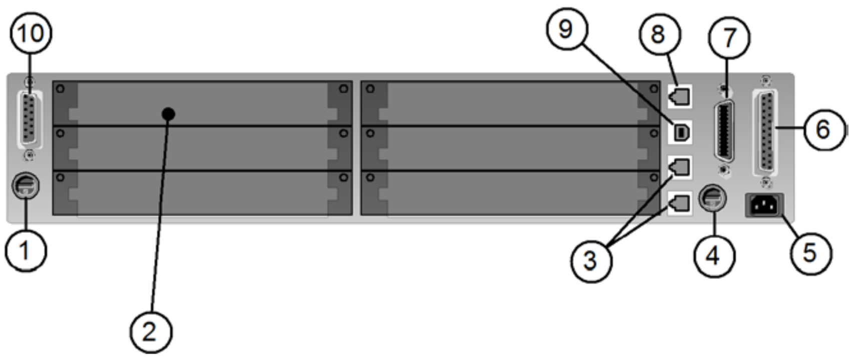
Front-panel channel attributes for channel settings

Front-panel channel setting	Symbol	Definition	Symbol meaning
DAC output enable settings	`Off`	Output enable is disabled	Used with 6 or 12 character label
	`On`	Output enable is enabled	Used with 6 or 12 character label

Rear panel summary

The following is a brief overview of the Series 3700A System Switch/Multimeter rear panel.

Figure 8: Rear panel features



Item	Description
1	Analog backplane AMPS fuse (on page 2-26)
2	Slots (on page 2-27) (6 slots)
3	TSP-Link connector (on page 2-27) (2 connectors)
4	Instrument fuse (on page 2-27)
5	Power connector (on page 2-27)
6	Digital I/O port (on page 2-28, on page 3-43)
7	GPIB connector
8	Ethernet connector
9	USB connector
10	Analog backplane connector (on page 2-32)

Rear panel connection details

The following topics describe how to connect the cable connections for the communication interfaces.

To properly set up the communications interfaces after connection, see the information in Communication interfaces (see "Remote communication interfaces" on page 2-53).

Analog backplane AMPS fuse

WARNING

For continued protection against fire hazard, replace fuse with same type and rating (3 A / 250 V). See Fuse replacement (on page A-2, on page 2-2) for details.

Slots

Use any of the six slots of the Keithley Instruments Series 3700A for switching cards. If a slot does not contain a card, make sure that you cover the slot with a slot cover.

To get information about an installed card, press the **SLOT** key.

NOTE

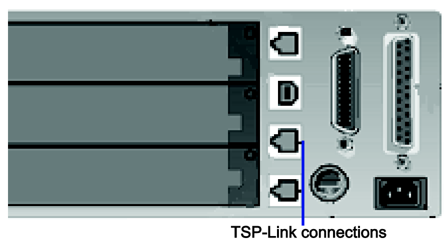
For complete information about Series 3700A switching cards, refer to the Series 3700A Switch and Control Cards Reference Manual (Keithley part number 3700AS-909-01) on the Product Information CD-ROM that came with your Series 3700A.

TSP-Link connector

Connect the TSP-Link connector to one of the TSP-Link connectors on the rear panel of the instrument.

The location of the TSP-Link connectors on the instrument are shown below.

Figure 9: Series 3700A TSP-Link connections



Instrument fuse

FOR CONTINUED PROTECTION AGAINST FIRE HAZARD, REPLACE FUSE WITH SAME TYPE AND RATING (1.25A / 250V). See Fuse replacement (on page A-2, on page 2-2) for details.

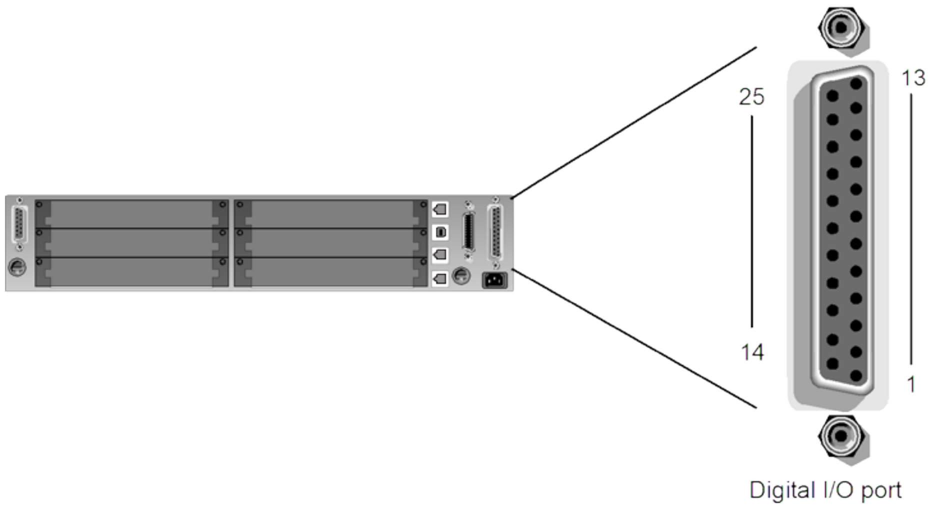
Power connector

Using the supplied line cord, connect to a grounded AC power outlet. See Line power connection for connection details.

Digital I/O port

The Series 3700A instruments have a digital input/output port that can be used to control external digital circuitry. For example, a handler that is used to perform binning operations can be used with a digital I/O port. The digital I/O port is a standard female DB-25 connector.

Figure 10: Digital I/O port



Pin	Description
1 ... 9	Digital I/O #1 ... Digital I/O #9
10 ... 14	Digital I/O #10 (high-current pins; see NOTE) ... Digital I/O #14
15 to 21	Ground
22	V EXT
23	V EXT
24	Pin reserved for future use
25	V EXT

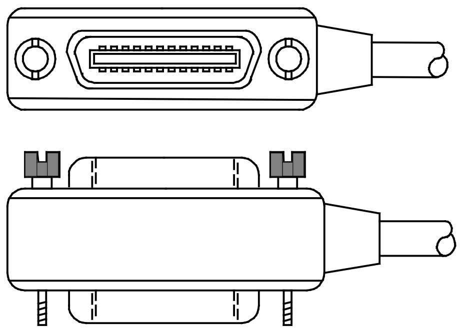
NOTE

For a schematic diagram of the digital I/O hardware, refer to the Series 3700A Specifications on the Keithley Instruments support website (*http://www.keithley.com/support*).
High-current pins (pins 10 to 14) can be used for binning applications or for external relays.

GPIB connector

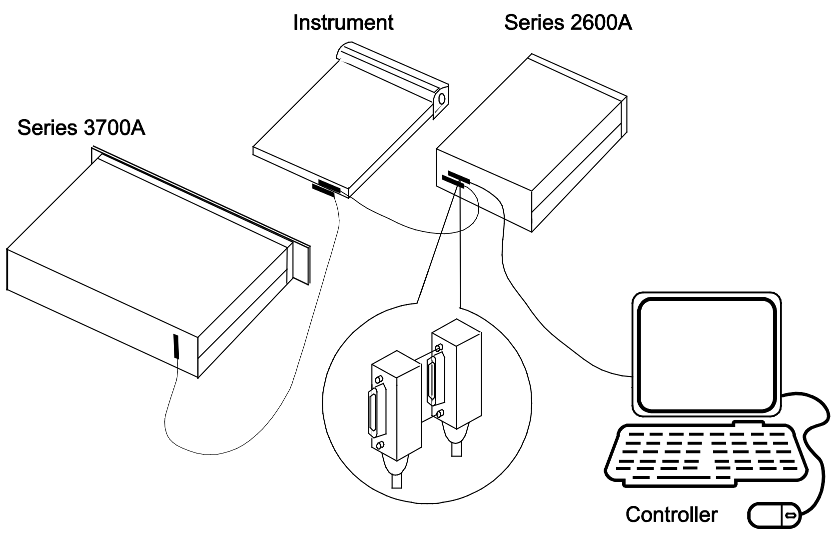
To connect a Series 3700A to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown below.

Figure 11: GPIB connector



To allow many parallel connections to one instrument, stack the connectors. Two screws are located on each connector to ensure that connections remain secure. The figure below shows a typical connection scheme for a multi-unit test system.

Figure 12: Series 3700A multiple parallel connections

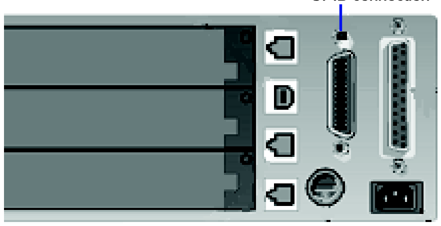


⚠ CAUTION

To avoid possible mechanical damage, stack no more than three connectors on any one unit. To minimize interference caused by electromagnetic radiation, use only shielded IEEE-488 cables. Contact Keithley Instruments for shielded cables.

To connect the Series 3700A to the IEEE-488 bus, line up the cable connector with the connector located on the rear panel. Install and tighten the screws securely, making sure not to overtighten them (the following figure shows the location of the connections).

Figure 13: Series 3700A GPIB connector



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

Ethernet connection

Connect the ethernet connector between the rear panel of the instrument and the host computer or network router. You can use an LAN crossover cable (RJ-45, male/male) or straight-through cable. The instrument automatically senses which cable you have connected.

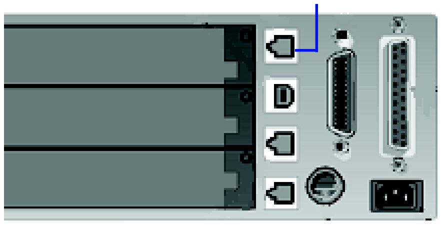
NOTE

The TSP-Link® connectors will accept a LAN connection, but will not be identified as a LAN and will not connect properly. Be sure to connect the LAN connector correctly.

Use this RJ-45 connector to connect the instrument to the local area network. When connecting directly to a computer, a crossover cable (included) must be used. When connecting to a network switch, router, or hub, a normal CAT-5 cable (not provided) should be used unless your equipment has Auto-MDIX capabilities. If it does have Auto-MDIX, the crossover cables may be used.

The figure below shows the location of the ethernet connector on the Series 3700A rear panel.

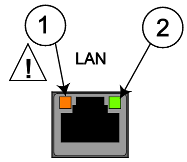
Figure 14: Series 3700A ethernet connection



LAN status LEDs

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

Figure 15: LAN Status



1 LED indicates the LAN port is connected to a 100 Mbps network
2 LED indicates the LAN port is connected to a 10 Mbps network

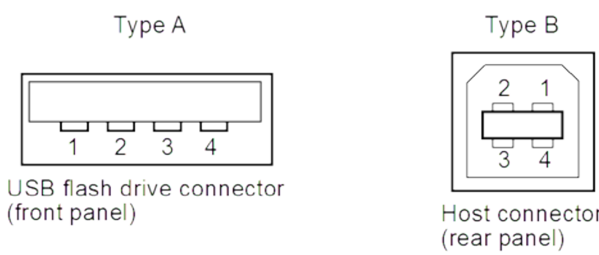
When the LED is:	The network:
Off	is not connected
On	is connected
Blinking	is sending or receiving data

USB connectors

The downstream USB-2.0 receptacle (Type B) located on the rear panel connects to a host. The front panel has an upstream USB-2.0 connector (Type A) that connects to a user supplied USB flash drive.

Use the rear connector to communicate with the instrument over USB by sending the desired commands. Use the front panel connector to insert a USB flash drive for saving or loading reading buffers, user setups, or scripts. See the Reference Manual for more information on reading buffers, user setups and scripts.

Figure 16: USB connectors

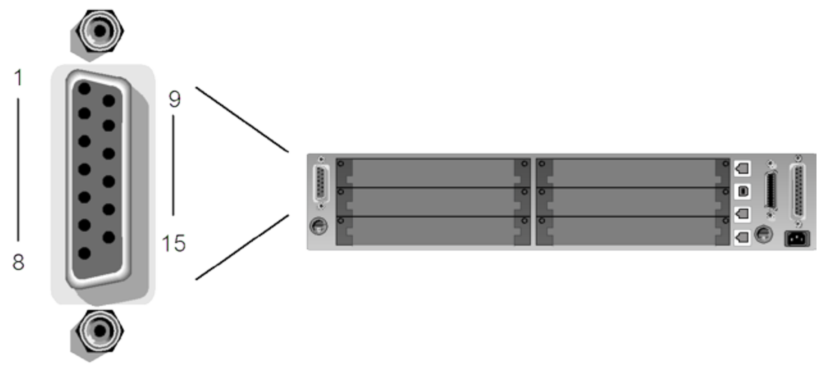


Pin number	Function
1	VBUS (5 volts)
2	D-
3	D+
4	Ground

Analog backplane connector

Refer to the following figure for analog backplane connector information. See Connections (see "Connection safety" on page 2-32) before making any connections.

Figure 17: Analog backplane connector



Analog backplane connector

The tables below contain pin numbers and descriptions for the analog backplane connector.

Description	Pin
Analog backplane 3-HI	5
Analog backplane 3-LO	6
Analog backplane 4-HI	7
Analog backplane 4-LO	8
Analog backplane 5-HI	12
Analog backplane 5-LO	13
Analog backplane 6-HI	14
Analog backplane 6-LO	15

Description	Pin
DMM-SLO	4
DMM-SHI	3
DMM-LO	2, 9
DMM-HI	1
AMP-LO	2, 9
AMP	10, 11

Connection safety

WARNING

Connection information for switching cards is intended for qualified service personnel. Do not attempt to connect DUT or external circuitry to a switching card unless qualified to do so.

To prevent electric shock that could result in serious injury or death, comply with these safety precautions:

Before making or breaking any connections to the switching card, make sure the instrument is turned off and power is removed from all external circuitry.

Do not connect signals that will exceed the maximum specifications of any installed switching card.

If both the rear analog backplane connector of the instrument and the switching card terminals are connected at the same time, the test lead insulation must be rated to the highest voltage that is connected. For example, if 300V is connected to the analog backplane connector, the test lead insulation for the switching card must also be rated for 300V.

Dangerous arcs of an explosive nature in a high energy circuit can cause severe personal injury or death. If the multimeter is connected to a high energy circuit when set to a current range, low resistance range, or any other low impedance range, the circuit is virtually shorted.

Dangerous arcing can result (even when the multimeter is set to a voltage range) if the minimum voltage spacing is reduced in the external connections. For details about how to safely make high energy measurements, see **High-energy circuit safety precautions** ***(on page 4-2).***

As described in the International Electrotechnical Commission (IEC) Standard IEC 664, the instrument is Installation Category I and must not be connected to mains.

Saved setups

You can restore the Series 3700A to one of six nonvolatile memory setup configurations (five user setups and one factory default), or to a setup stored on an external USB flash drive. As shipped from the factory, the Series 3700A powers up with the factory default settings, which cannot be overwritten. The default settings are also in the five user setup locations, but may be overwritten. The factory default settings are listed in the command descriptions in the TSP command reference (on page 8-1).

The setup configuration that is used when the instrument powers up can be changed.

Saving user setups

You can save the present Series 3700A setup to internal nonvolatile memory or a USB flash drive.

To save a user setup to nonvolatile memory from the front panel:

1. Configure the Series 3700A to the settings that you want to save.
2. Press the **MENU** key.
3. Select **SETUP** and then press the **ENTER** key.
4. Select the **SAVE** menu item and then press the **ENTER** key.
5. Select **INTERNAL** and then press the **ENTER** key.
6. Select the user number (1 through 5), and press the **ENTER** key.

To save a user setup to an external USB flash drive from the front panel:

1. Configure the Series 3700A to the settings that you want to save.
2. Insert the USB flash drive into the USB port on the front panel of the Series 3700A.
3. Press the **MENU** key.
4. Select **SETUP** and then press the **ENTER** key.
5. Select **SAVE** and then press the **ENTER** key.
6. Select **USB**. The file name `setup000.set` is displayed.
7. Turn the navigation wheel to change the last three digits of the file name and then press the **ENTER** key.

Recalling a saved setup

You can recall setups from internal nonvolatile memory or a USB flash drive at any time.

To recall a saved setup from the front panel:

1. Press the **MENU** key to access the main menu.
2. Select **SETUP**, and then press the **ENTER** key.
3. Select the **RECALL** menu item, and then press the **ENTER** key.
4. Select one of the following:
   - INTERNAL
   - USB

   USB only: Select the appropriate file and then press the **ENTER** key.

Start-up configuration

You can specify the Series 3700A start-up (power-on) configuration from the front panel. Set the start-up configuration to a previously stored setup (recalled from internal nonvolatile memory).

To select the power-on setup:

1. Press the **MENU** key to access the main menu.
2. Select **SETUP**, and then press the **ENTER** key.
3. Select **POWERON**, and then press the **ENTER** key.
4. Select the configuration you want to use on startup.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key to return to the main menu.

Saving user setups from a remote interface

Saving and recalling user setups

Use the `setup.save()` and `setup.recall()` functions to save and recall user setups.

To save and recall user setups using remote commands:

The following example saves the present setup as setup 1, and then recalls setup 1:

```
-- Save the present setup to nonvolatile memory.
setup.save(1)
-- Recall the saved user setup from nonvolatile memory.
setup.recall(1)
```

Restoring the factory default setups

Use one of the reset functions to return the Series 3700A to the original factory defaults. An example of each type of reset is shown in the following program examples.

Restore all factory defaults of all nodes on the TSP-Link® network:

```
reset()
```

Restore all factory defaults (note that you cannot use `*rst` in a script):

```
*rst
```

Restore all factory defaults:

```
setup.recall(0)
```

Restore all channels on all slots to defaults:

```
channel.reset("allslots")
```

Reset just the local TSP-Link node:

```
localnode.reset()
```

Start-up (power-on) configuration

You can specify the Series 3700A start-up (power-on) configuration. Use the `setup.poweron` attribute to select which setup to return to upon power-up. To set the `setup.poweron` configuration attribute:

```
setup.poweron = n       -- Select power-on setup.
```

Where:

n = `0` (`*RST`/`reset()` factory defaults)

n = `1`

Using the web interface

Introduction

The Series 3700A web interface can be used with your choice of web browsers, including Microsoft® Internet Explorer®, Mozilla® Firefox®, Google Chrome™, and Apple® Safari®. Using the web interface, you can review instrument status, control the instrument, and upgrade the instrument over a LAN connection.

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

All examples in this manual can be run through the TSB Embedded (on page 2-48) web application that is available on the instrument web interface.

Connect to the instrument web interface

To connect to the instrument web interface, you must have an LAN connection from the computer to the instrument. See LAN concepts and settings (on page B-1) for specific connection instructions.

The web interface requires the web browser plug-in Sun Java™ Runtime Environment Version 6 or higher. Installation files are available from http://www.java.com/en/download/manual.jsp (*http://www.java.com/en/download/manual.jsp*).

The ActiveX control and Java applets are installed from the instrument but, depending on the browser security settings, they may require the users permission to be downloaded and installed.

After the instrument is connected and Java is installed, to connect to the instrument:

1. Open an internet browser, such as Windows Internet Explorer (v6.0 or higher only).
2. In the Address box, enter the IP address of the instrument (to find the IP address, from the front panel of the instrument, select **MENU** > **LAN** > **STATUS** > **IP-ADDRESS**).

The Home page of the instrument web interface is displayed.

Web interface home page

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

- The instrument model, serial number, firmware revision, and LXI information
- A list of slots and the switch cards that are installed in each slot
- An **ID** button to help you locate the instrument
- Links to the instrument web options, including TSB Embedded.

Identify the instrument

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

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

To identify the instrument:

In the upper right corner of the Home page, click 

.

The button turns green 

, and the LAN status indicator on the instrument blinks.

Click 

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

Log in to the instrument

The web interface has both interactive and read-only pages. These pages are always listed in the navigation panel on the left side of the web interface. You can review information on any of the pages without logging in, but to change information, you must log in.

Pages that contain information you can change include a **Login** button. Once you have logged in to one page of the web interface, you do not need to log in again unless you reload the page.

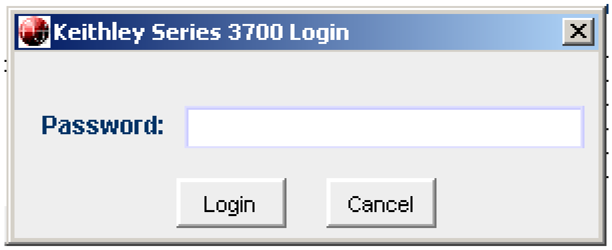
To log into the instrument:

1. Open a page that contains a **Login** button, such as one of the Cards pages, Scan Builder, or TSB Embedded.

Figure 18: Web interface login

2. Click **Login**. The login dialog box is displayed.
3. Enter the password (the default is **admin**).

Figure 19: 3700A Enter web interface password



4. Click **Login**.

NOTE

The default password is **admin**. If the password has been changed, it is available from the front panel of the instrument. Press **MENU** > **LAN** > **STATUS** > **PASSWORD**.

Card pages

The card pages are interactive pages where you can work with channels in each slot.

To open a card page, on the left navigation, click the slot number.

There is a specific page for each card installed in the mainframe. The page displays a grid that shows the relay configuration of the switch card.

Figure 20: Series 3700A web interface Cards page

Open and close slots from the card pages

You can open and close channels from the card pages in several ways.

The simplest method is to click a connection. The channel changes state to open or closed. When the channel is open, the connection will look similar to one of the following graphics (the actual item on the web interface depends on the installed card):

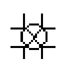
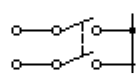


Figure 21: Series 3700A web interface relay open



When the channel is closed, the connection will look similar to one of the following:

Figure 22: Web interface closed channel

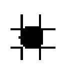
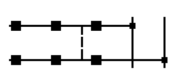


Figure 23: Web interface relay closed



To specify the type of close, select a Channel Action Type from the box in the upper right before closing a channel. The options are:

- **Channel Close**: Close the selected channel without affecting the state of any other channels.
- **Exclusive Slot Close**: Close the selected channel and open any closed channels in the same slot.
- **Exclusive Close**: Close the selected channel and open any closed channels in the instrument (the only closed channel is the selected channel).

You can open all channels in a slot by clicking **Open Slot**.

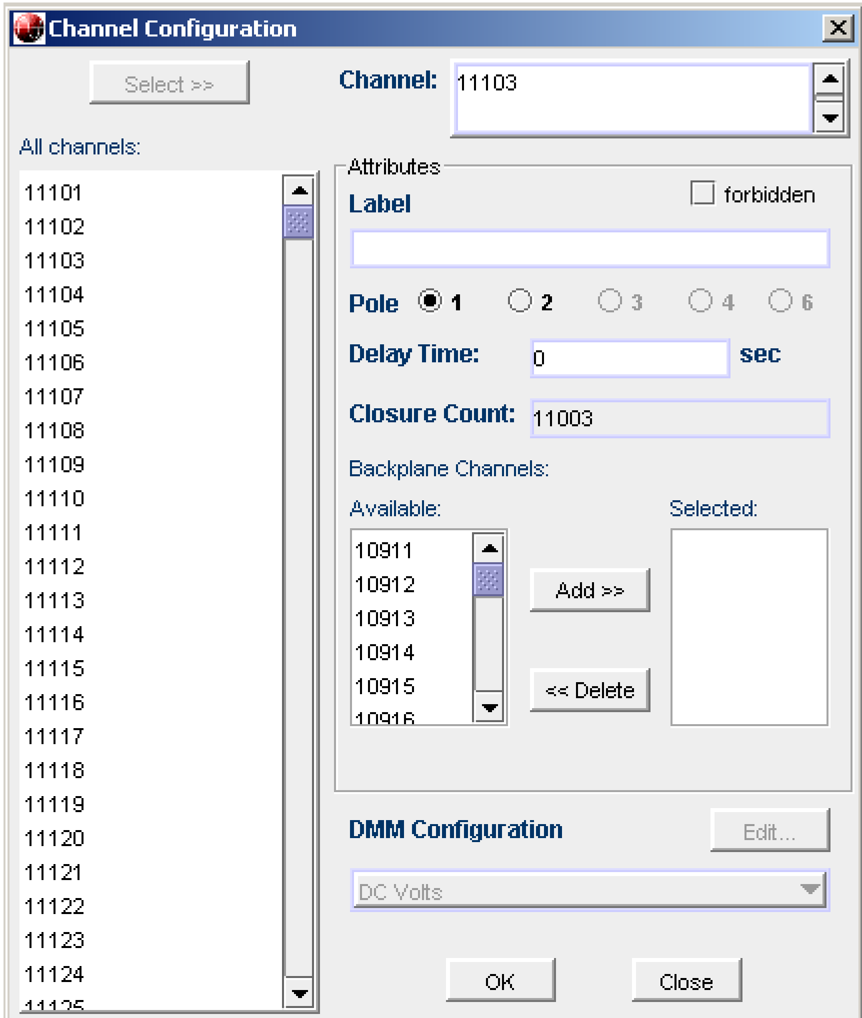
You can open all channels in the instrument by clicking **Open All**.

For more information on opening and closing channels, see "Working with channels" in the Series 3700A Reference Manual.

Configure channels from the web interface

To configure channels from the web interface, right-click the channel. The Channel Configuration dialog box is displayed.

Figure 24: Series 3700A channel configuration dialog box



In this dialog box, you can set:

- **Label:** The label for the channel. This is the same as the command `channel.setlabel()`.
- **Forbidden:** Select this box to set the channel to forbidden. This prevents the channel from being closed from any interface. Note that if the channel is used in a channel pattern, the pattern is deleted when you set the channel to forbidden to close. An analog backplane relay can be marked as forbidden to close.
- **Pole**: Pole setting for multiplexer (MUX) channels indicates if the paired MUX channel should be included when performing a close or open operation on channel.
  In a switching module that has 60 channels, the Series 3700A automatically pairs Channels 1 through 30 with Channels 31 through 60 (respectively) when the pole setting for a channel is set to 4-pole. Once you configure the pole setting of a switching channel for 4-pole, the associated paired channel becomes unavailable for switching operation. For example, if 3003 is set to 4-pole and its paired channel is 3033, you cannot set attributes or perform close or open operations on channel 3033. If you specify channel 3303 for a close or open operations, an error code -221, "Paired channel settings conflict," is generated.
  Matrix channels have fixed pole settings. Multiplexer channels pole settings may be changed.

- **Delay Time**: The additional delay to incur after the relay settles when closing the channel. Enter the value for the delay in seconds. The total delay for channel close is this delay plus the relay settling time.
- **Backplane channels:** You can select the backplane relay with this option.
- **DMM Configuration:** Click Edit to set up configuration of the DMM for this channel.

This dialog box also displays the closure count. See Determining the number of relay closures (on page 2-92) for information.

Set up channel patterns from the web interface

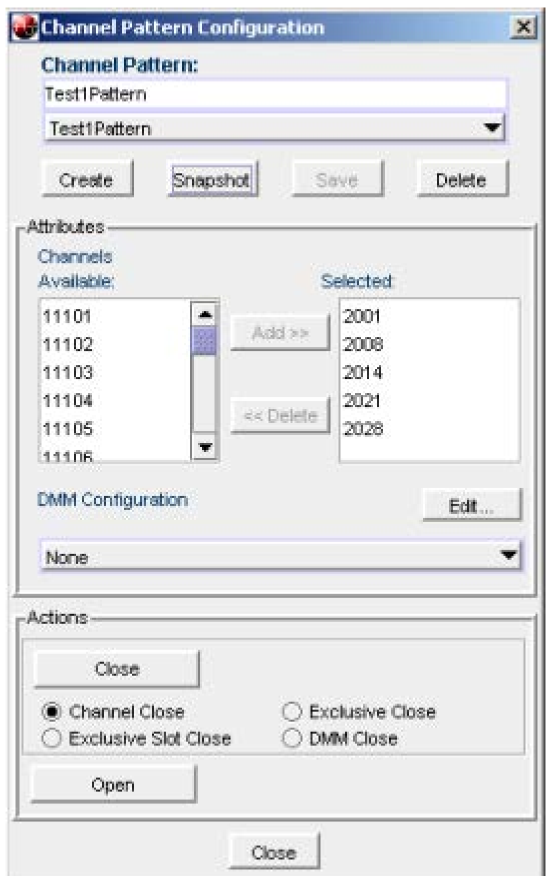
You can use channel patterns as a convenient way to refer to a group of switching channels and backplane relays with a single alphanumeric name. When you perform close or open operations on a channel pattern, only the channels and analog backplane relays that are in the channel pattern are affected.

There is no speed difference when performing close and open operations on channel patterns compared to performing the same operations on individual channels or a list of channels.

To create a channel pattern from the web interface:

1. From the left navigation, click a slot.
2. Click **Pattern** (above the Channel Action Type box). The Channel Pattern Configuration dialog box is displayed.

Figure 25: Series 3700A Channel pattern configuration dialog box



3. Enter a name in the box at the top.
4. From the Channels Available list, select the channels you want to add. You can use Ctrl+click and Shift+click to select multiple channels.
5. Click **Add**. You can add as many channels as needed.
6. Click **Create**.

To create a channel pattern from the web interface using the Snapshot feature:

1. Close the channels that you want to include in the pattern.
2. Click **Pattern** (above the Channel Action Type box). The Channel Pattern Configuration dialog box is displayed.
3. Enter a name in the box at the top.
4. Click **Snapshot**. A new pattern is created that contains the closed channels.

To delete a channel pattern from the web interface:

1. Select the name of the pattern that you want to delete.
2. Click **Delete**.

For more information regarding patterns, including opening and closing the channels that are in patterns, see Channel patterns (on page 2-96).

Reset a slot from the web interface

You can reset all the relays in the displayed slot by clicking **Reset Slot**.

When you reset the relays in a slot:

- Any closed channels and analog backplane relays open
- The poles of all channels reset to 2-pole operation and paired channels are changed to match
- Labels return to default of slot, channel or slot, row, column
- Analog backplane relays specified by the `channel.setbackplane()` function are cleared
- Delays are set to zero
- If the channel is forbidden to close, it is cleared from being forbidden to close
- The DMM configurations of all channels are set to `nofunction`
- If any of the slot's channels are in channel patterns, the patterns are deleted

The rest of the instrument settings are unaffected.

Scan Builder page

The Scan Builder page allows you to set up and run scans and triggers.

A scan is a series of steps that opens and closes switches sequentially for a selected group of channels. During each step, actions occur, such as waiting for a trigger, taking a measurement on an external instrument, and completing a step count. Scans automate actions that you want to perform consistently and repeatedly on a set of channels.

Triggers are events that prompt the instrument to move from one step to another in a scan. Triggers can come from a variety of sources, such as a key press, digital input, or expiration of a timer. The sequence of actions and events that occur during the scan is called the trigger model, described in Trigger model (on page 3-2).

Scanning and triggering allow you to synchronize actions across channels. You can set up a scan using the trigger model to precisely time and synchronize the Series 3700A between channels and multiple instruments. You can also use triggers without the triggering model to set up a scan to meet the needs of a specific application that does not fit the triggering model.

NOTE

If you use Scan Builder to create a scan, use the options in the Scan Builder page to run the scan. Using the TSB Embedded page may not give you the expected results.

Create a scan list

Before you can run a scan, you must create a scan list. A scan list is a set of steps that runs in order during a scan. Each step contains a channel, channels, or channel patterns that you want to measure in that step. Each step is acted on separately during the scan.

You can mix channel patterns and individual channels in a scan list. Note that the steps are executed in the order in which they are added to the scan.

NOTE

Before setting up a scan list, make sure your channels and channel patterns are configured. See Working with channels (on page 2-89) for detail.

If you change the channel configurations or channel patterns after the scan list has been set up, you may not see expected results. If the change prevents the scan from functioning properly (such as deleting something referenced by the scan), an error message is logged.

To create a scan list from the web interface:

1. From the left navigation of the web interface Home page, select **Scan Builder**.
2. In the Add Channel By list on the right, select **Number** to add the channels individually or **Pattern** to select patterns. You can include both channels and patterns in the same scan list.
3. If you selected **Number**, select the channel numbers from the list. To remove your selections from the Add Channel By list, click **Clear Channel Selection**. You can use Ctrl+click to select multiple channels and Shift+click to select a range of channels.
4. If you selected **Pattern**, select a pattern from the Channel Pattern list.
5. Click **Add Step**. The channels and patterns are added to the Steps list.
6. In the **Scan Count** box, enter the number of times you want to repeat the steps in the scan.
7. In the **Measure Count** box, enter the number of times you want to repeat the measurement in the scan.
8. Under **Use DMM Configuration**, select "assigned to the channel(s)" to use the DMM configurations that are assigned to the channels, or select "selected from below to override" to choose from a list of DMM configurations. Repeat these steps as needed to build the scan steps. The scan is saved as you build it.

Clear the scan list from the web interface

Clearing the scan list deletes all channels and channel patterns from the scan list.

To clear the scan list from the web interface:

1. From the left navigation area of the web interface home page, select **Scan Builder**.
2. Click **Scan Clear**.

Review the scan list

You can review the existing scan list to see which channels and channel patterns are listed, and in which order.

To review the scan list from the web interface:

1. From the left navigation of the web interface Home page, select Scan Builder.
2. Select the **Build & Run** tab. The scan list is shown in the Steps box.

Reset the scan list

You can clear the scan list and return scan settings to their factory defaults using scan reset. A scan reset does not affect any settings in the instrument except the scan list and trigger model.

The settings that are affected are:

- Channels and patterns are removed from the scan list
- Bypass: Returned to default setting of ON
- Mode: Returned to default setting of Open All
- Scan count: Returned to default setting of 1
- Trigger to start scan: Set to Immediate
- Trigger to continue channel action for each scan step: Setting is cleared
- Arm (Scan Start Stimulus) is set to None
- Channel Action Stimulus is set to Channel Ready Event
- Channel Ready Event is set to None
- Scan Complete Even is set to None

To reset the scan list from the web interface:

1. From the left navigation of the web interface Home page, select Scan Builder.
2. Click **Scan & Trigger Reset**.

Run the scan

You can run a scan in one of several ways:

- **Background:** Runs the scan in the background so that you can perform other tasks while the scan is running. You can use the Query State to check scan status.
- **Step by Step:** Steps through the scan.

To run the scan as a background scan from the web interface:

Click **Execute Background** or **Step by Step**.

Stop the scan

To stop the scan from the web interface:

On the Build & Run tab, click **Abort**.

Monitor the state of the scan

To monitor the state of the scan, you can click **Query State** on the Build & Run tab. **Query State** displays the current state of the scan, which can be:

- Empty: No scan defined
- Building: Scan list is being created
- Running: Scan in process
- Success: Scan completed successfully

Set up simple triggers

You can set up triggers to control your scan using the options on the Simple Trigger tab. You can set:

- The event that starts the scan
- The time interval event that controls the channel action for each step of the scan
- The time interval event that controls how measurements are taken during the scan

To see these options, click the **Simple Trigger** tab from the top of the Scan Builder page.

Selecting triggers

You can choose the triggers that will be used to start the scan. The options to start the scan are:

Immediate: When Immediate is selected, the scan starts as soon as you click **Execute Background** on the Build & Run tab. Select Immediate when you do not have trigger requirements that must be met before the scan starts. This is the default selection.

Digital Input: When selected, you select the digital line (1 to 14) that is used to start a scan. You can select falling or rising for the digital input. Falling selects the falling edge trigger. Rising selects the rising edge of the trigger.

NOTE

If Other is displayed in the mode list, a different mode (other than falling or rising) is already selected. Other is not a mode and cannot be selected. It is only an indicator that the digital triggering is already set up for a different mode. See the Series 3700A Reference Manual, "Using the web interface" section, and the "Advanced triggering" topic for other options.

Time: When selected, you can select options that define when the scan starts and at what rate triggers will occur.

You can select the trigger to use to continue channel action for each scan. The options to continue channel action are:

Immediate: When immediate is selected, the scan immediately steps to the next channel in the scan list. This is the default setting.

Digital Input: When selected, you select a digital line (1 to 14) that is used to trigger the instrument to step to the next channel. You can select falling or rising for the digital input. Falling selects the falling edge trigger. Rising selects the rising edge of the trigger.

Every *N* seconds: This parameter adds a fixed delay between each channel. The delay occurs before the next channel in the scan list is closed.

You can also select select the trigger to use to take a measurement for each scan step.

Immediate: When immediate is selected, the measurement is taken as soon as the channel is closed. This is the default setting.

Digital Input: When selected, you select a digital line (1 to 14) that triggers the instrument to take a measurement. You can select falling or rising for the digital input. Falling selects the falling edge trigger. Rising selects the rising edge of the trigger.

Every *N* seconds: This parameter adds a fixed delay after the channel is closed and before the measurement is taken.

Advanced triggering

The Advanced Trigger tab of the Scan Builder allows you to set the options that are available from the Simple Trigger tab, as well as more sophisticated options to control scan triggering.

The Advanced Trigger tab uses the trigger model flowchart to help you visualize and define the input and output triggers to the scan.

For more information on the trigger model, see Trigger model (on page 3-2).

The options on the Advanced Trigger tab include:

- **Mode**: Select Open All to open all slots before the scan starts. Select Open Selective to open only channels that are involved in scanning; closed channels that are not involved in scanning remain closed. Select Fixed ABR to open all channels involved in the scan, but close all required backplane relays before the scan.
- **Arm (Scan Start) Stimulus**: Select the event that causes the arm event detector trigger to be set to the detected state (the scan can begin).
- **Measure Stimulus**: Select the event that causes the measurement event detector to be set to the detected state (the measurement can begin).
- **Measure Complete Event To:** Select the recipient of the Measurement Complete Event.
- **Channel Ready Event To**: Select the recipient of the Channel Ready Event.
- **Scan Complete Event To**: Select the recipient of the Scan Complete Event.

There is also a **Config** button available for each of the options except Mode. When you click this button, a dialog box with additional options for the selection is displayed.

Set the scan mode

The scan mode determines how channels are opened before the start of the scan.

You can select:

- **Open all:** All slots are opened.
- **Open select:** All channels selected in the scan list are opened; any closed channels remain closed if they are not in the scan list.
- **Fixed ABR**: All necessary backplane relays are closed before the scan.

To set the scan mode from the web interface:

1. Select the **Advanced Trigger** tab.
2. Select **Mode**.
3. Select **Fixed ABR**, **Open All** or **Open Selective**.

DMM web page

The DMM web page allows you to configure the various DMM functions, create user-defined DMM configurations and reading buffers, and take measurement readings that may or may not be stored in a reading buffer.

At the top of the DMM web page is a visual representation of the actual front-panel reading and indicators (note that the indicators are not in the same location as the front panel, and not all are on the page). When enabled, you will see DMM display indicators such as `REL`, `FILT`, `MATH`, and `AUTO` (see (2) The display for a complete list of these indicators and their meanings). Below this front-panel view is a scrolling list of the active DMM settings, including the active function and its supporting attribute values. Scroll the list to view the values set.

Use the **Change Active Settings** button to change the settings for the active function. When clicked, a dialog box appears that contains settings that can be changed for that function, and you can also change the function. Changing functions adjusts the displayed attributes to be correct for the newly selected function. After making you desired changes, click the **Close** button at the bottom of the page.

Use the **Edit Configurations...** button to select a DMM configuration from a drop-down list. Once the DMM configuration is selected, the dialog box shows the settings that can be changed for that configuration. This dialog box is very similar to the the dialog box for changing active settings. Until a user-defined DMM configuration is saved with the **Save as** button, only the factory default DMM configurations exist in the list.

After making changes to a factory default DMM configuration, click the **Save as** button to open a dialog box that allows you to name the new configuration and save it to use later. Once you have entered and saved your DMM configuration, select it from the pull-down menu to make changes to it. After making changes, click the **Save as** button again; by default it uses the same name to overwrite your user-defined configuration with the updated settings.

In the Reading Buffer area, you can create a reading buffer or select an existing reading buffer to store your readings, or you can select None. With None selected, readings are not stored in a reading buffer. The readings are only displayed in the Data Table area on the web interface. You can clear the data table (click the **Clear Table** button), save data to to a computer (click the **Save Table to PC** button), select the timestamp format for time associated with the readings (select Relative, Seconds, Time or Full). If you want to store readings in a buffer, but no buffers exist, click the **Create** button. Once the buffer is created, it becomes the selected buffer to store readings.

When a new buffer is selected (including None) for readings, the storage the Data Table gets cleared.

When data is being stored to a reading buffer, the readings show on the front-panel display, but do not appear in the Data Table on the web interface until you click the **Refresh Data Table** button. If you want to view the data graphically, click the **View/Refresh Chart** button. If you want to save the reading buffer data to the USB flash drive on the instrument, click the **Save to USB** button.

Click the **Measure** button to take a single reading, or click the **Loop Measure** button to take a measurement per seconds configured below the button. Set the seconds between measurements by using the slide bar below to select ranges from 0.5 to 20 seconds. The measurements will automatically appear in the Data Table when the reading buffer is configured to None. If the reading buffer isn't set to None, click the **Refresh Data Table** to see the measurements.

TSB Embedded

TSB Embedded is an application that includes a command line interface that you can use to issue ICL commands. It also offers script-building functionality. TSB Embedded resides in the instrument.

Script management options

Existing scripts are listed in the User Scripts box on the left side of TSB Embedded.

To run a script, click the name of the script and then click **Run**.

To delete a script, click the name of the script and click **Delete**. The script is deleted from the User Scripts list and from the nonvolatile memory of the instrument.

To stop operation of a script, click **Abort Script**.

To export the selected script to a flash drive, click **Export Script to USB**. Place a flash drive in the USB port on the front panel of the instrument. In TSP, enter the name as appropriate and click **OK**. Scripts are saved to a file with the extension `tsp`. TSP files are native to Test Script Builder or TSB Embedded, but they can be opened and edited in any text editor.

To import scripts from the computer, click **Import from PC**. Select the directory that contains the file. You can only import files with the extension `tsp`.

To clear the name box and the box that contains the script, click **Clear**.

To view the contents of a script, type the name of a script in the TSP Script box and click **View Script**.

To create a script, see Create a script using TSB Embedded (on page 2-49).

Command line interface

Console: Enter command line entries here to send commands to the instrument. Click **Enter** to send the command. The commands and output are shown in the Instrument Output box.

To resend a command, click the button at the left of the Console box.

Instrument control

To reset the entire TSP-enabled system, including the controlling node and all subordinate nodes, click **Reset**.

Create a script using TSB Embedded

NOTE

If you are using TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

You can create a script from the instrument web page with TSB Embedded. When you save the script in TSB Embedded, it is loaded into the run-time environment and saved in the nonvolatile memory of the instrument. For information about using TSB Embedded, select the **Help** button on a web page or the Help option from the navigation pane on the left side of the web interface.

To create a script using TSB Embedded:

1. In the TSP Script box, enter a name for the script.
2. In the input area, enter the sequence of commands to be included in the script.
3. Click **Save Script**. The name is added to the User Scripts list on the left.

Admin page

Through the Admin page, you can change the instrument password and the instrument time.

Change the password

To change the password for the web interface:

1. In the web interface, from the left navigation, click **Admin**. A login page is displayed.
2. Enter the current password in the Password box. (The default is "admin".)
3. Click **Login**. The Administration page is displayed.
4. In the Current Password box, enter the current password.
5. In the New Password box, enter the new password.
6. In the Confirm New Password box, enter the new password again.
7. Click **Submit**. The new password takes effect immediately.

Change the instrument date and time

To change the date and time of the instrument:

1. In the web interface, from the left navigation, click **Admin**. A login page is displayed.
2. Enter the current password in the Password box (the default is "admin").
3. Click **Login**. The Administration page is displayed.
4. Enter the Year.
5. Select the Month, Day, Hour, Minutes and Seconds from the lists.
6. Click **Submit**. The new time and date information takes effect immediately.

Unit page

Save: Save the setup of the instrument.

Recall: Recall the setup of the instrument that was saved with the Save button.

Create Config Script: Save the set up of the instrument as a script.

To create the script:

1. Click **Create Config Script**. The Create Config Script dialog box is displayed.
2. To create a script that will run automatically when the instrument is powered on, select "Auto-execute on powerup." Note that this will overwrite the existing autoexec script.
3. To create a script with a new name, select Name and enter the name.
4. Click **OK**.

Reset: Resets all instruments in the TSP-enabled system. This is only available if the instrument is the master.

Open All: Click this to open all relays on all slots.

Upgrade Firmware: Select a firmware upgrade file to download to the instrument and begin the upgrade process.

Channel Connect Rule: Select the channel connect rule. See Connection methods for close operations (on page 2-89) for detail.

Digital I/O Lines: This is the tool to configure the 14 digital I/O lines of the instrument. Values can be read or written to the ports, or each individual bit can be toggled. "Write Protect" can be set individually for any I/O line.

Generate Report: This generates an instrument report you can use to:

- Review card or instrument information, including a basic description, the firmware revision, and the serial number.
- Review configuration information, including card configuration, DMM configuration, calibration information, and number of poles.
  - Review the number of closures for each channel on the selected slots.
- The number of closures are the closures that have occurred over the lifetime of the card.

To print the report, click **Print**.

To clear the report information from the screen, click **Clear**.

LXI page

The Series 3700A is a LXI Class B instrument. The LXI page is a read-only page that displays the LXI information about the instrument.

IP Config

The IP Config allows you to review and change the LAN connection information.

See Change the IP configuration through the web interface for more information.

Log page

The event log records all LAN[0-7] triggers generated and received and can be viewed over any command interface, including the web interface. The following figure shows the view of the LAN[0-7] event log from the embedded web interface.

Up to 32 LAN[0-7] events are logged and shown on this page. The event log is circular and rolls over after 32 events are captured. The LAN[0-7] events correspond to the lan.trigger[1-8] subsystem.

Figure 26: Event log

The timestamp, event identifier, the IP address and the domain name identify the incoming and outgoing LXI trigger packets. The following table provides detailed descriptions for the columns in the event log.

Event log descriptions

Column title	Description	Example
Receive Time	Displays the date and time when the LAN trigger occurred in UTC, 24-hour time	06:56:28.000 8 May 2008
EventID	Identifies the lan.trigger[N] that generates an event	`LAN0 = lan.trigger[1]` `LAN1 = lan.trigger[2]` `LAN2 = lan.trigger[3]` `LAN3 = lan.trigger[4]` `LAN4 = lan.trigger[5]` `LAN5 = lan.trigger[6]` `LAN6 = lan.trigger[7]` `LAN7 = lan.trigger[8]`
From	Displays the IP address for the device that generates the LAN trigger	localhost 192.168.5.20
System Timestamp	A timestamp that identifies the time the event occurred. The timestamp uses the following: PTP timestamp Seconds Fractional seconds The Series 3700A does not support the IEEE-1588 standard; the values in this field are always 0 (zero)	
HWDetect	Identifies a valid LXI trigger packet	LXI
Sequence	Each instrument maintains independent sequence counters: One for each combination of UDP multicast network interface and UDP multicast destination port. One for each TCP connection.	
Domain	Displays the LXI domain number (the default value is 0 (zero))	0 1523
Flags	Contain data about the LXI trigger packet	Values: 1 - Error 2 - Retransmission 4- Hardware 8 - Acknowledgments 16 - Stateless bit
Data	The Series 3700A does not support the IEEE-1588 standard; the values in this field are always 0 (zero)	

Remote communication interfaces

This section shows you how to connect instruments to the following remote communication interfaces:

- Universal serial bus (USB)
- Local area network (LAN)
- General purpose interface bus (GPIB or IEEE-488)

The USB can be used for single ASCII-based commands.

The LAN

It describes how to configure and troubleshoot these interfaces on computers with Windows 2000, Windows XP, Windows Vista, and Windows 7 operating systems.

It also describes the I/O software, drivers, and application software that can be used with Keithley's instruments.

Supported remote interfaces

The Model 3706A supports the following remote interfaces:

- GPIB. General purpose interface bus is an IEEE-488 instrumentation data bus.
- LAN. Local area network (LAN) communications provide the flexibility to build scalable and functional test or data acquisition systems with a large degree of flexibility.
- USB. Communicate with the instrument over a Type B USB connection.

NOTE

The Model 3706A can be controlled from only one communication interface at a time. The first interface on which it receives a message takes control of the instrument. It ignores the other interfaces until the instrument is returned to local operation.

For more information about the remote interfaces, see:

- GPIB setup (on page 2-58)
- LAN concepts and settings (on page B-1)
- USB communications (on page 2-53)

USB communications

To use the rear-panel USB connection, you must have the Virtual Instrument Software Architecture (VISA) layer on the host computer. See How to install the Keithley I/O Layer (on page 2-70) for more information.

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

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

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

NOTE

The full version of National Instruments (NI®) VISA provides a utility to create a USB driver for any other kind of USB device that you want to communicate with VISA. For more information, see the NI VISA website (see National Instruments VISA site - *http://www.ni.com*).

Communicate with the instrument

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

USB[board]::manufacturer ID::model code::serial number[::USB interface number][::INSTR]

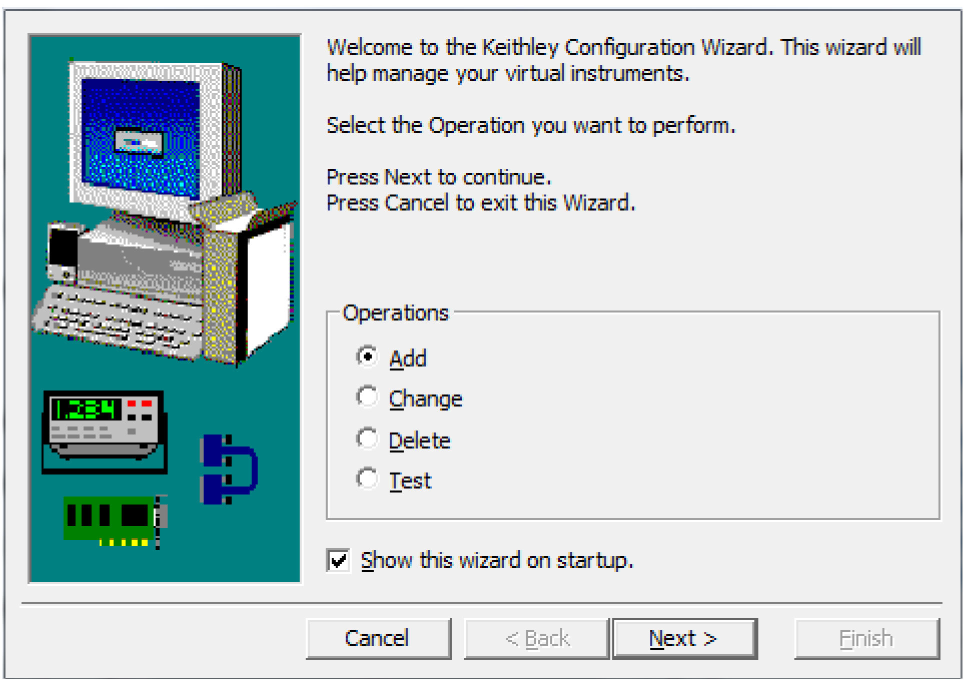
This requires that you determine the parameters. You can gather this information by running a utility that automatically detects all instruments connected to the computer.

If you installed the Keithley I/O Layer, the Keithley Configuration Panel is available from the Microsoft® Windows® Start menu in the Keithley Instruments menu.

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

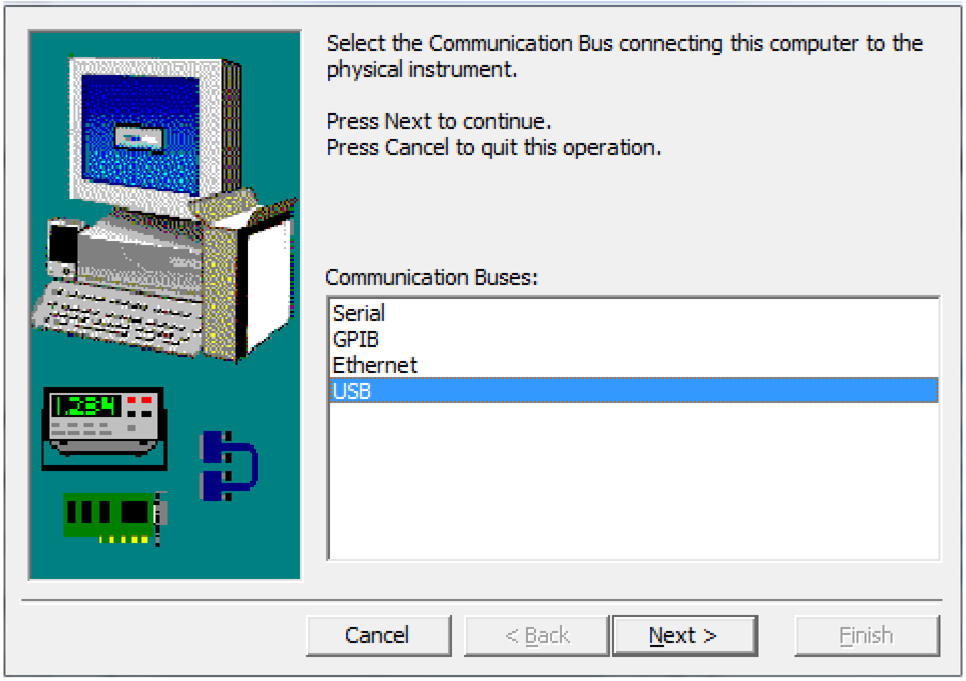
1. Start the Keithley Configuration Panel. The Select Operation dialog box is displayed.
2. Select **Add**.

Figure 27: Select Operation dialog box



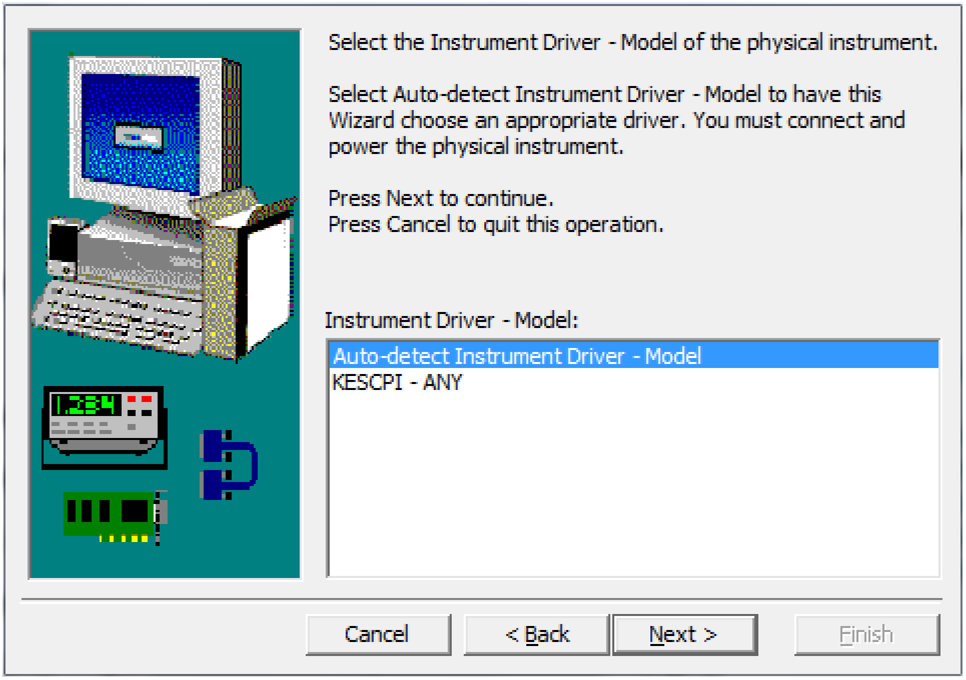
3. Click **Next**. The Select Communication Bus dialog box is displayed.

Figure 28: Select Communication Bus dialog box



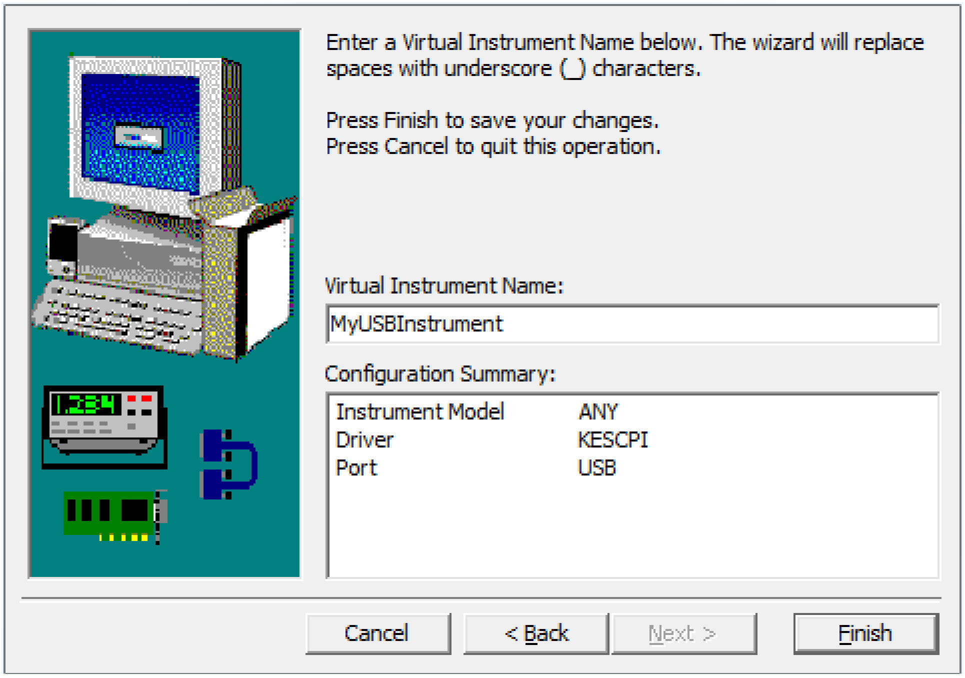
4. Select **USB**.
5. Click **Next**. The Select Instrument Driver dialog box is displayed.

Figure 29: Select Instrument Driver dialog box



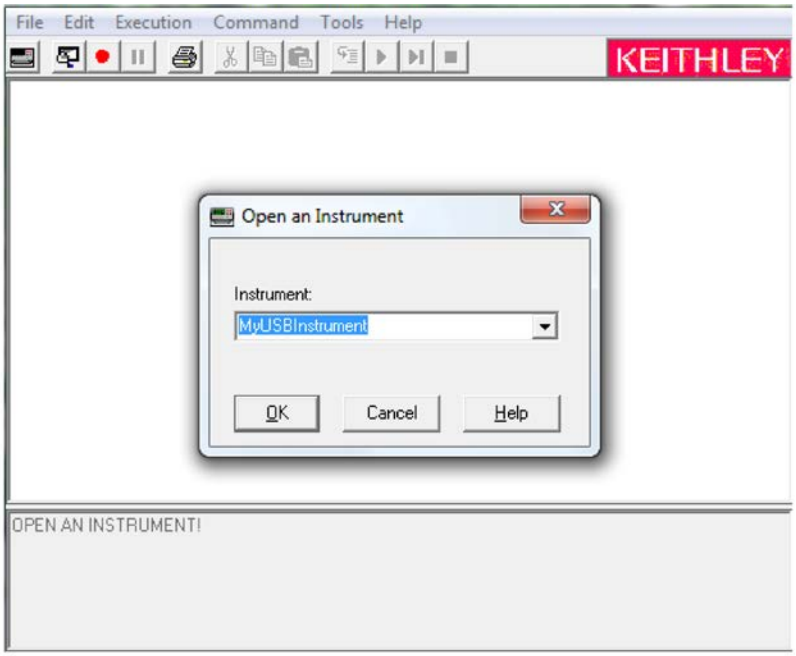
6. Select **Auto-detect Instrument Driver - Model**.
7. Click **Next**. The Configure USB Instrument dialog box is displayed with the detected instrument VISA resource string displayed.
8. Click **Next**. The Name Virtual Instrument dialog box is displayed.

Figure 30: Name Virtual Instrument dialog box



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

Figure 31: Keithley Communicator Open Instrument



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

NOTE

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

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

Additional USB information

This section provides further details and more advanced information about the USB bus and test-and-measurement instruments.

Connecting multiple USB instruments to the computer

The most convenient way to connect USB instrumentation to the computer is to plug a USB cable directly from the instrument to the computer. If you have more than one USB instrument or have other USB devices, such as printers, keyboards, and mice, you might not have enough USB connectors on the computer.

To gain more ports, you can use a USB hub or add more USB controller cards if you have available PCI or PCI Express slots.

There are two types of USB hubs that you can use with Series 3700A:

- **Bus powered**: This type of hub draws its power from the USB bus and can only supply 100 mA (USB 2.0) per port.
- **Self powered**: This type of hub has an external power supply and can supply up to 500 mA per port (USB 2.0).

GPIB setup

This section contains information about GPIB standards, connections, and address selection.

NOTE

The GPIB connector is optional and may not be present on your instrument.

GPIB standards

The GPIB is the IEEE-488 instrumentation data bus, which uses hardware and programming standards originally adopted by the Institute of Electrical and Electronic Engineers (IEEE) in 1975. The instrument is IEEE Std 488.1 compliant and supports IEEE Std 488.2 common commands and status model topology.

Install the GPIB driver software

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

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

Install the GPIB cards in your computer

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

Set the GPIB address

The GPIB address value is set to 16 at the factory. The address can be set to any address value between 0 and 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

To change the GPIB address:

1. Press the **MENU** key.
2. Select **GPIB > ADDRESS**. Press the navigation wheel ◷ to display the current address.
3. Choose the appropriate GPIB address.
4. Press **ENTER** to save the address.

The address value is saved in nonvolatile memory and will not change when a reset() (on page 8-317) command is sent or when the power is turned off and then turned on again.

When the GPIB bus is operating, you can use the gpib.address (on page 8-261) attribute to change the GPIB address remotely.

Enable GPIB

By default, the instrument is set to GPIB enabled. You only need to enable it if GPIB control was disabled.

To enable control through the GPIB:

1. Press the **MENU** key.
2. Select **GPIB**. Press the navigation wheel ◷ to display the GPIB MENU.
3. Select **ENABLE**. Press the navigation wheel ◷.
4. To enable GPIB, select **ON**. To disable it, select **OFF**.
5. Press **ENTER** to save the setting.

You must turn the instrument on and off before the setting takes effect.

Communicate with instruments

The GPIB driver software you installed installs a interactive dumb terminal program that allows you to send commands to the instrument. They directly call the GPIB driver support libraries.

For the KPCI-488LPA and KUSB-488B GPIB controller from Keithley Instruments, the configuration utility is called the KI-488 Diagnostic Tool. It is available from the Windows Start menu at **Keithley Instruments > KI-488 > KI-488 Diagnostic Tool**.

For the KUSB-488A GPIB controller from Keithley Instruments, the configuration utility is called TrTest. It is available from the Windows Start Menu at **Keithley Instruments > GPIB-488-CEC > TrTest**.

For National Instruments GPIB controllers, you can use NI-MAX. Start NI-MAX. If your hardware is installed correctly, you should see the controller in the GPIB section of the tree control on the left side. Select it and right-click to see an option to communicate with the instrument.

NOTE

If you want to use the GPIB controller with instrument driver (such as VXIPnP or IVI) or high-level software, you must also install I/O software, which installs the VISA layer. See How to install the Keithley I/O Layer (on page 2-70).

Terminator

When receiving data over the GPIB, the instrument terminates messages on any line feed character or any data byte with EOI asserted (line feed with EOI asserted is also valid). When sending data, it appends a line feed character to all outgoing messages. The EOI line is asserted with the terminating line feed character.

However, if you want your program to communicate with all I/O buses on the instrument (GPIB, USB, LAN (VXI-11 and raw socket)), it is good practice to add a line feed to the end of the outgoing command. Use VISA and the same program will work with all the I/O buses by changing the resource string in the VISA Open method.

Front-panel GPIB operation

This section describes aspects of the front panel that are part of GPIB operation, including messages, status indicators, and the LOCAL key.

Error and status messages

The front-panel display may show error and status messages (see Displayed error and status messages). See Error summary list for a list of status and error messages that are associated with IEEE-488 programming.

LOCAL key

The EXIT (LOCAL) key cancels the remote state and restores local operation of the instrument. Pressing the EXIT (LOCAL) key also turns off the REM indicator and returns the display to normal if a user-defined message was displayed.

If the LLO (Local Lockout) command is in effect, the EXIT (LOCAL) key is also inoperative. Note that pressing the EXIT (LOCAL) key will also abort any commands or scripts that are being processed.

GPIB reference

General bus commands

General commands are commands that have the same general meaning, regardless of the instrument (for example, DCL). The following table lists the general bus commands.

General bus commands

Command	Effect on Series 3700A
REN	Goes into remote operation when next addressed to listen. See REN (on page 2-61) for details.
IFC	Goes into talker and listener idle states. See IFC (on page 2-61) for details.
LLO	**LOCAL** key locked out. See LLO (on page 2-61) for details.
GTL	Cancel remote; restore Series 3700A front-panel operation. See GTL (on page 2-61) for details.
DCL	Returns the Series 3700A and all devices on the GPIB to known conditions. See DCL (on page 2-62) for details.
SDC	Returns the Series 3700A to known conditions. See SDC (on page 2-62) for details.
GET	Initiates a trigger. See GET (on page 2-62) for details.
SPE, SPD	Serial polls the Series 3700A. See SPE, SPD (on page 2-62) for details.

REN

The remote enable (REN) command is sent to the Series 3700A by the controller to set up the instrument for remote operation. Generally, the instrument should be placed in the remote mode before you attempt to program it over the bus. Setting REN true does not place the instrument in the remote state. You must address the instrument to listen after setting REN true before it goes into remote operation.

IFC

The interface clear (IFC) command is sent by the controller to place the Series 3700A in the talker idle state and the listener idle state. The instrument responds to the IFC command by canceling front-panel TALK or LSTN lights, if the instrument was previously placed in one of these states.

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

LLO

When the instrument is in remote operation, all front-panel controls are disabled, except the LOCAL and OUTPUT OFF keys (and the POWER switch). The local lockout (LLO) command disables the LOCAL key, but does not affect the OUTPUT OFF control, which cannot be disabled.

GTL

Use the go to local (GTL) command to put a remote-mode instrument into local mode. Leaving the remote state also restores operation of all front-panel controls.

DCL

Use the device clear (DCL) command to clear the GPIB interface and return it to a known state. Note that the DCL command is not an addressed command, so all instruments equipped to implement DCL will do so simultaneously.

When the Series 3700A receives a DCL command, it:

- 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 does not affect instrument settings and stored data.

SDC

The selective device clear (SDC) command is an addressed command that performs essentially the same function as the device clear (DCL) command. However, because each device must be individually addressed, the SDC command provides a method to clear only selected instruments, instead of clearing all instruments simultaneously with the DCL command.

When the Series 3700A receives an SDC command, it:

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

An SDC does not affect instrument settings and stored data.

GET

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

SPE, SPD

Use the serial polling sequence to obtain the Series 3700A serial poll byte. The serial poll byte contains important information about internal functions (see Status model (on page 6-18, on page D-1, "*Status Byte Register overview*" on page D-4)). Generally, the serial polling sequence is used by the controller to determine which of several instruments has requested service with the SRQ line. The serial polling sequence may be performed at any time to obtain the status byte from the Series 3700A.

Configure the GPIB controllers

Each instrument on a GPIB bus needs a unique address from a range of 0 to 30. Generally, the GPIB host controller is on address 0. However, there are GPIB controllers that adopt the address of 21. To be safe, do not configure any of the instruments for 21 or 0.

If you do need to change the host controller address, consult the controller documentation.

For the KPCI-488LPA and KUSB-488B GPIB controller from Keithley Instruments, the configuration utility is called the KI-488 Diagnostic Tool. It is available from the Microsoft® Windows® Start menu at **Keithley Instruments > KI-488 > KI-488 Diagnostic Tool**.

For the KUSB-488A GPIB controller from Keithley Instruments, the configuration utility is called GPIB Configuration. It is available from the Windows Start Menu at **Keithley Instruments > GPIB-488 > GPIB Configuration**.

For National Instruments (NI™) GPIB controllers, you can use NI-MAX. Start NI-MAX. If your hardware is installed correctly, you will see the controller in the GPIB section of the tree control on the left side. Select it and right-click to see an option to configure the controller. Do not forget to save your settings.

GPIB status indicators

The remote (REM), talk (TALK), listen (LSTN), and service request (SRQ) indicators show the GPIB bus status. Each of these indicators is described below.

REM

This indicator is illuminated when the instrument is in the remote control state. When the instrument is in the remote control state, all front-panel keys, except for the EXIT (LOCAL) key, are locked out. When REM is off, the instrument is in the local control state and front-panel operation is restored.

TALK

This indicator is on when the instrument is in the talker active state. Place the instrument in the talk state by addressing it to talk with the correct talk command. TALK is off when the instrument is in the talker idle state. Place the instrument in the talker idle state by sending a UNT (untalk) command, addressing it to listen, or by sending the IFC (interface clear) command.

LSTN

This indicator is on when the Series 3700A is in the listener active state, which is activated by addressing the instrument to listen with the correct listen command. LSTN is off when the instrument is in the listener idle state. Place the instrument in the listener idle state by sending UNL (unlisten), addressing it to talk, or by sending the IFC (interface clear) command over the bus.

SRQ

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

LAN communications

This section provides an overview of LAN communications for the Series 3700A. For detailed information about setting up your LAN interface, refer to LAN concepts and settings (on page B-1).

You can communicate with the instrument using a local area network (LAN).

When you connect using a LAN, you can use a web browser to communicate with the instrument through the instrument's internal web page and other web applets.

Series 3700A are class B LXI version 1.3 compliant. They are scalable test instruments with direct connections to host computers. They can also interact with a DHCP or DNS server and other LXI compliant instruments on a LAN.

The Series 3700A are compliant with IEEE standard 802.3 (Ethernet) and support full connectivity on a 10 Mbps or 100 Mbps network.

NOTE

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

Overview of LAN instruments

When Ethernet ports became standard on computers, it was logical that instrumentation would follow. The VXI-11 protocol, which was standardized on in the early 1990s, is the standard used to emulate GPIB over Ethernet.

Even though Ethernet became the standard LAN technology on instruments, LAN instruments from different vendors differed in the approach they took. Some vendors only supported static IP, whereas others had DHCP, DLLA (Auto-IP), and static addressing. The LXI consortium was started to standardize what should be in all instruments that conform to LXI.

An instrument that conforms to LXI version 1.3 must have the following:

- All three IP addressing modes: DHCP, Auto-IP, and static IP.
- A web server that has some standard Ethernet configuration parameters:
  - IP configuration: IP address, subnet mask, gateway.
  - Password protection on anything that might change the instrument state.
  - A control on the web page that flashes an LED or some form of indicator on the front panel of the instrument. LXI calls this the Device Identification Functionality. This allows you to identify the web page you are currently looking at with the instrument. This helps you identify a specific instrument in a rack of similar model instruments.

- A reserved URL in the instrument that provides an xml document that has standard configuration information. This can be useful for software tools that need to identify the instruments and their capabilities. The URL is **http://<host>:<port>/lxi/indentification**.
- An IVI driver for the instrument.
- A LAN Status (fault) indicator.
- VXI-11 discovery protocol.
- LAN reset button or menu option. LXI calls this the LAN Configuration Initialize (LCI).

  When the LXI-defined LAN reset is selected, the instrument reverts its LAN settings to a known set of defaults. The default LAN settings for LXI instruments are:

  - DHCP and Auto-IP enabled. LXI refers to this as the Auto IP address mode (compared to the manual address mode, which is fixed or static IP addressing).
  - Web password is reset to the factory default.
  - Ping responder enabled.
- Dynamic DNS and mDNS enabled.

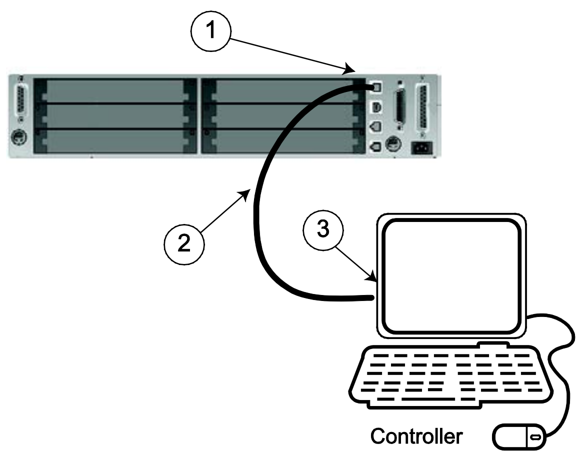
LXI Version 1.3 added the requirement of mDNS (multicast DNS) discovery.

LAN cable connection

The Series 3700A includes two Model CA-180-3A cables (LAN crossover cables). One cable is for the TSP-Link® network and the other cable is for LAN communication.

Use the following figure as a guide when making LAN connections.

Figure 32: Series 3700A LAN connection



1 Series 3700A ethernet port (LAN)

2 Straight-through LAN cable or crossover LAN cable (Model CA-180-3A)

3 Ethernet port (located on the host computer)

Supplied software

The majority of software applications and all instrument drivers from Keithley Instruments depend on some, or all, of the following software components:

- NI-VISA™
- VISA shared components
- IVI shared components
- NI™ CVI™ runtime engine
- NI™ IVI™ compliance package
- Keithley instrument driver
- LabVIEW™ driver
- Test Script Builder (TSB) Add-in
- J2SE™ Runtime Environment

These software components are included on the CD-ROMs that came with your instrument, and are also available for download at the Keithley Instruments support website (*http://www.keithley.com/support*).

Instrument driver types

There are several different styles of instrument drivers. Keithley Instruments provides three different instrument drivers for the Series 3700A: A native LabVIEW driver, an IVI-C driver, and an IVI-COM driver. You need to pick the style that best suits the application development environment (ADE) that you are using. For example, if you are using LabVIEW, you would pick a native LabVIEW driver. If a native LabVIEW driver is not available then you can use an IVI-C driver as LabVIEW has the option of creating a wrapper for the IVI-C driver.

LabVIEW supports IVI-COM drivers but they are definitely not the first or second choice. However, if it is the only driver type for the instrument, it can be used.

If LabWindows/CVI or C/C++ is your programming language, an IVI-C driver is the best option. For VB6 and any .NET language (C#, VB.NET, and so on), an IVI-COM driver would be the best option.

Sometimes instrument vendors do not provide all three driver types. Most languages can accommodate other driver types, but this is not optimal.

The following sections describe the different driver types in more detail.

VXIPnP drivers

VXI (Vixie) plug-and-play (VXIPnP) style drivers are Win32 DLLs that have some standard functions defined by the VXIPnP Alliance, such as:

- init
- close
- error_message
- reset
- self_test
- Read
- Initiate
- Fetch
- Abort

The application programming interface (API) was defined so that users of instruments would have a familiar API from instrument to instrument. There are some basic guidelines when creating APIs for your instrument, such as using VISA data types and how to construct the CVI hierarchy.

LabVIEW drivers

Native LabVIEW drivers

A native LabVIEW™ driver is a LabVIEW driver that is created using entirely built-in LabVIEW VIs — it does not make any calls to external DLLs or Library files. This makes the driver portable to all the platforms and operating systems that LabVIEW and VISA supports (currently, Linux® on x86, Mac OS® X, and Microsoft® Windows®).

National Instruments (NI™) maintains a native LabVIEW driver style guide (*http://zone.ni.com/devzone/cda/tut/p/id/3271*).

LabVIEW driver wrappers

All IVI-C drivers have a function panel file (.fp) that shows a hierarchy of the function calls into a DLL. It is a tool that guides a user to select the correct function call in the driver, since a DLL only has a flat API entry point scheme (unlike COM or .NET). Any CVI-generated .fp file can be imported into LabVIEW and LabVIEW will generate a wrapper for the DLL. The drawback here is that the driver is dependent on the DLL, which is not portable and is therefore Windows-specific.

Obtaining instrument drivers

To see what drivers are available for your instrument:

1. Go to the Keithley Instruments support website (*http://www.keithley.com/support*).
2. Enter the model number of your instrument.
3. Select Software Driver from the list.

For LabVIEW™, you can also go to National Instrument's website and search their instrument driver database.

Instrument driver examples

All Keithley drivers come with examples written in several programming languages that show you how to do the most common things with the instruments.

Install the driver. The examples are in the Microsoft® Windows® Start menu, under **Keithley Instruments > Model Number** (where Model Number is the instrument model number).

IVI shared components

The IVI shared components are a similar concept to the VISA shared components. The IVI Foundation provides class drivers for:

- All the supported instruments (DMM, Scope, Fgen, and so on)
- The configuration store

The IVI shared components also create the installation folders and registry keys that all IVI drivers and support files use for installation.

Interchangeable Virtual Instruments (IVI) style drivers

The major problem with VXIPnP drivers was that the API was not specific to the instrument. For something as standard as measuring DC volts on a digital multimeter (DMM), it would be a good idea if there were a set of standard functions to do this.

The IVI Foundation (*http://www.ivifoundation.org*) defined a set of application programming interfaces (APIs) for the following instruments: DMM, function generator, DC power supply, scope, switch, spectrum analyzer, RF signal generator and power meter. They are currently working on class APIs for some other instrument types.

There are two types of IVI drivers: IVI-COM drivers use Microsoft® COM technology to expose driver functionality, while IVI-C drivers use conventional Microsoft® Windows® DLLs to export simple C-based functions.

For more information about IVI drivers and the differences between the COM, C, and .NET interfaces, see Making the Case for IVI (*http://pacificmindworks.com/docs/Making%20the%20Case%20for%20IVI.pdf*).

NI CVI runtime engine

IVI-C drivers that are created using National Instruments (NI™) LabWindows/CVI environment depend on either the CVI runtime (cvirte.dll), or the instrument support run-time (instrsup.dll), and must be present on the system for them to run.

NI IVI Compliance Package

The National Instruments (NI™) IVI Compliance Package is a software package that contains IVI class drivers and support libraries that are needed for the development and use of applications that leverage IVI instrument interchangeability. The IVI Compliance Package also is based on and is compliant with the latest version of the instrument programming specifications defined by the IVI Foundation.

The NI ICP installer installs the IVI shared components, CVI runtime engine, and the instrument support runtime engine.

Keithley I/O layer

The Keithley I/O Layer (KIOL) is a software package that contains several utilities and drivers. It is mainly used as a supplement to IVI drivers, or application software like Test Script Builder (TSB).

The KIOL contains:

- NI-VISA™ Runtime-Time Engine
- Keithley Configuration Panel
- Keithley Communicator

NI-VISA Runtime

NI-VISA™ is National Instruments (NI™) implementation of the VISA standard. There are two versions. The full version contains diagnostic and configuration tools such as NI-Spy and NI-MAX and the binary run-time-only files. The run-time version contains only the binary files (DLLs) that allow the drivers to operate.

The Keithley I/O Layer (KIOL) contains a licensed version of the NI-VISA runtime.

If you already have NI software (such as LabVIEW or LabWindows) installed, you have a valid license that can be used with Keithley drivers and application software.

If you do not have NI software installed, to use Keithley drivers or application software, you must install the KIOL. This installs a valid, licensed copy of the NI-VISA runtime to use with Keithley drivers or application software. KIOL installs a valid license for the NI-VISA Run-Time Engine only (not the full version of NI-VISA).

Keithley Configuration Panel

The Keithley Configuration Panel is a configuration utility for IVI drivers, similar to NI-MAX. It also has the ability to autodetect USBTMC instruments and LAN instruments that support the VXI-11 protocol.

Keithley Communicator

The Keithley Communicator is a dumb terminal program that uses VISA to communicate with the instrument.

Computer requirements for the Keithley I/O Layer

The Keithley I/O Layer version C02 supports the following operating systems:

- Microsoft® Windows® (32-bit & 64-bit) Business with Service Pack 1 or later
- Microsoft Windows Vista® Business (32-bit & 64-bit) with Service Pack 2 or later
- Windows XP Professional (32-bit) with Service Pack 3 or later
- Windows 2000 Professional with Service Pack 4 plus update KB891861 or later

Note that Windows 95, Windows 98, Windows ME, Windows NT, Windows XP (64-bit) operating systems are not supported.

How to uninstall previous versions of the Keithley I/O Layer

If you have an earlier version of the Keithley I/O Layer software installed on your computer, you must uninstall it.

To uninstall the Keithley I/O layer:

1. From the Control Panel, select **Add/Remove Programs**.
2. Uninstall the following components:
   - Keithley I/O Layer
   - Keithley I/O Layer Suite
   - Keithley SCPI-based Instrument IVI-C Driver
   - NI-VISA Run-Time Engine x.x.x (if present) (x.x.x is the VISA version)
1. Reboot your computer.

How to install the Keithley I/O Layer

NOTE

Before installing, it is a good idea to check the Keithley Instruments website (*http://www.keithley.com*) to see if a later version of the Keithley I/O Layer is available. On the website, select the **Support** tab, under **model number**, type KIOL, and select **Software Driver**.

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

The software installs the following components:

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

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

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

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

1. Download the Keithley I/O Layer Software from the Keithley Instruments website (*http://www.keithley.com*) as described in the note. The software is a single compressed file and should be downloaded to a temporary directory.
2. Run the downloaded file from the temporary directory.
3. Follow the instructions on the screen to install the software.
4. Reboot your computer.

Special installation considerations

Situations may occur during installation that cannot be handled automatically by the installation utility. The installation utility will warn you if one of these situations is detected. The sections below describe the action you must take before the installation can be completed.

Mismatch between IVI Shared Components and IVI Engine Detected

The IVI Shared Components and IVI Engine are software components that may be installed by various test and measurement software applications, instrument drivers, and so on. Keithley I/O Layer software requires that these components, if present, be compatible versions. The installation utility will detect a mismatch, which must be corrected before the software installation can proceed. If this situation is detected, the Keithley I/O Layer software installation will automatically stop.

The recommended way to resolve this situation is to install the IVI Compliance Package (ICP) software from National Instruments (NI™). You may download the ICP software and release notes from National Instrument's website. When the ICP installation is complete, restart the Keithley I/O Layer software installation.

Non-National Instruments VISA detected

VISA software is used to communicate with the instrument and may be installed by various test and measurement software applications, instrument drivers, and so on. Keithley I/O Layer software requires and will install National Instruments NI-VISA™ software. The installer will detect if another vendor's version of VISA is already installed on the computer. If this occurs, the installer will pause and display a warning message. The warning message displays the vendor of the detected VISA in its title bar, if this can be determined. Make a note of the vendor name. At this point, you may elect to continue the installation, which will overwrite the existing VISA installation with NI-VISA. This will allow the Keithley I/O Layer software to operate properly, but may cause other applications or instrument drivers that were dependent on the existing VISA to malfunction.

The recommended way to resolve this situation is to perform the following steps:

1. Exit the Keithley I/O Layer software when the warning message is displayed. Make note of the VISA vendor in the warning message (if any).
2. Uninstall the non-NI VISA software.
3. Uninstall Tektronix VISA by selecting OpenChoice TekVISA from the Control Panel Add/Remove programs list.
4. Uninstall Agilent VISA by selecting Agilent I/O Libraries Suite from the Control Panel Add/Remove programs wizard list.

5. Uninstall other versions of VISA by selecting the appropriate entry from the Control Panel Add/Remove Programs Wizard list.
6. Restart the Keithley I/O Layer software installation.
7. If the pre-existing version of VISA was supplied by Tektronix or Agilent (as displayed in the warning message), you may safely reinstall that version of VISA once Keithley I/O Layer software installation is complete. When you reinstall Tektronix or Agilent VISA, it may prompt you to preserve the current VISA version, which you should do. This will usually restore the operation of any dependent applications or drivers.
8. If the pre-existing version of VISA was supplied by a vendor other than Tektronix or Agilent, we recommend that you do not reinstall it, because this will likely cause the Keithley I/O Layer software to malfunction.

Installation troubleshooting

If problems occur during installation, it might be helpful to install the components individually. Errors messages might appear that will help you resolve the installation issue.

If problems occur during installation:

1. Follow the instructions to uninstall all the KIOL components in Special installation considerations (on page 2-71).
2. Rerun the KIOL installer. Note where the installer unpacks the files (usually in a temporary folder).
3. Cancel the installer.
4. Go to the folder where the files were unzipped.
5. Run the setup.exe for each of the following components in the following order:
   - IVI Compliance Package (ICP)
   - NI-VISA™ Run-Time Engine
   - KIOL
   - Keithley SCPI Driver
1. Ignore all the other folders.
2. Reboot the computer.

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

The Keithley I/O Layer interconnects many other installers.

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

- National Instruments NI™ IVI Compliance Package
- National Instruments NI-VISA™ Run-Time Engine
- IVI Shared Components
- Visa Shared Components
- Keithley SCPI Driver

After uninstalling components, reboot the computer.

Addressing instruments with VISA

VISA allows you to communicate with the instrument on different communication buses by changing a resource string that gets passed in with the viOpen function, in VISA-C, or with the Open method on the VISA-COM resource manager object.

For detailed information about the format of the resource string, refer to the VISA specification VPP4.3 at the IVI Foundation website, or refer to the help file provided by the vendor of the VISA implementation you are using.

The following sections describe the resource strings for some of the communication types that Keithley supports. Any field that has [] (square brackets) around it is optional and will revert to a default value.

Addressing instruments through the LAN

VISA supports two different LAN protocols, each of which has a different resource string.

VXI-11 is a protocol that emulates GPIB over the LAN. Series 3700A supports this protocol. The resource string is:

```
TCPIP[board]::host address[::LAN device name][::INSTR]
```

board is the network interface card in the computer. This value is usually skipped and VISA determines the correct network interface card (if you have more than one) by looking at the IP address.

host address can be either a valid DNS hostname, mDNS hostname, or the IPv4 IP (only) address of the instrument.

LAN device name is a method of addressing secondary instruments at the main IP address, similar to secondary addressing on the GPIB bus. The default is `inst0`.

A **raw socket** connection requires more work by the driver or application program to make sure the correct amount of data has been sent or received correctly. All Keithley instruments support the raw socket connection.

```
TCPIP[board]::host address::port::SOCKET
```

The *board* and the *host address* are the same as for the VXI-11 protocol.

port is the port to which to connect on the instrument. For the Series 3700A, the port is 5025. See Instrument LAN protocols for a complete list of port numbers.

Addressing instruments using USB

```
USB[board]::manufacturer ID::model code::serial number[::USB interface
   number][::INSTR]
```

`board` is not used (`0`).

`manufacturer ID` is the USB.org reserved four-digit hexadecimal code for the instrument vendor company. Keithley Instruments hexadecimal code is `0X5E6`.

`model code` is the model number of the instrument. For example, when addressing a Model 3706A, use `0X3706`.

`serial number` is the serial number of the instrument.

`USB interface number` identifies which USBTMC interface on the instrument to address (usually `0`).

NOTE

Also see USB VISA identifiers.

Addressing instruments through GPIB

There are two different resource classes in VISA for the GPIB bus.

INSTR is the basic class that everyone uses. It allows application software to send and receive data and commands without dealing with some low level GPIB nuances. This class is recommended for typical GPIB communication.

The **INTFC** class allows finer control over the GPIB controller card in the computer. You must comply with the IEEE-488.1 protocol and tell the instrument to listen and the controller to talk before sending a message to the instrument. This class allows you to communicate to the instrument using low-level GPIB commands. Refer to your VISA documentation for more details on how to use this class.

The GPIB INSTR resource class format is:

```
GPIB[board]::primary address[::secondary address][::INSTR]
```

`board` is the number of the GPIB card, if there are more than one in the computer. If there is only one GPIB card, don not include `board`, but do not leave a space.

`primary address` is the main GPIB address of the instrument, which can be changed, if necessary, through the front panel of the instrument.

`secondary address` is for secondary addressing in GPIB. Some instruments have subinstruments or cards inside the main instrument or backplane. The primary address identifies the main instrument. The secondary address identifies subinstruments. Refer to the instrument user manual for the secondary address, if applicable.

Sending raw commands to an instrument

The next sections show you how to use VISA-C and VISA-COM to send raw instrument commands without using the instrument drivers.

VISA-C sample code

The following is a simple C/C++ console application that reads back the instrument identification string using VISA-C. You need to include visa.h and link with the visa32.lib file.

```
#include "stdafx.h"
#include <visa.h>

#define checkErr(fCall)      if (error = (fCall), (error = (error <
    0) ? error : VI_SUCCESS)) \
                                 {goto Error;}  else error = error

int _tmain(int argc, _TCHAR* argv[])
{
      ViSession defaultRM, vi;
      char buf [256] = {0};
      ViStatus  error = VI_SUCCESS;

      /* Open session to GPIB device at address 22 */
      checkErr(viOpenDefaultRM(&defaultRM));
      checkErr(viOpen(defaultRM, "GPIB0::14::INSTR", VI_NULL,VI_NULL, &vi));

      /* Initialize device */
      checkErr(viPrintf(vi, "*RST\n"));

      /* Send an *IDN? string to the device */
      checkErr(viPrintf(vi, "*IDN?\n"));
      ViUInt16 status = 0;
      do
      {
      checkErr(viReadSTB(vi, &status));
      printf("ReadSTB = %X\n", status);
      } while(status == 0);

      /* Read results */
      checkErr(viScanf(vi, "%t", &buf));
      /* Print results */
      printf ("Instrument identification string: %s\n", buf);

      /* Close session */
      checkErr(viClose(vi));
      checkErr(viClose(defaultRM));

Error:

      if(error < VI_SUCCESS)
         printf("Visa Error Code: %X\n", error);
      printf("\nDone - Press Enter to Exit");
   getchar();

      return 0;
}
```

VISA-COM sample code

This example gets the instrument identification string using VISA-COM in C#.

The first thing to do is add a reference to the VISA-COM interop DLL, which is usually located at C:\Program Files\IVI Foundation\VISA\VisaCom\Primary Interop Assemblies\Ivi.Visa.Interop.dll.

```
using Ivi.Visa.Interop;

namespace WindowsApplication1
{

      public class IdnSample: System.Windows.Forms.Form
      {
private Ivi.Visa.Interop.FormattedIO488 ioDmm;
      //
   }
   }
      private void IdnSample_Load(object sender, System.EventArgs e)
      {
         ioDmm = new FormattedIO488Class();

         SetAccessForClosed();
      }

      private void btnInitIO_Click(object sender, System.EventArgs e)
      {

       try
          {
         ResourceManager grm = new ResourceManager();
               ioDmm.IO = (IMessage)grm.Open("GPIB::16::INSTR",
                  AccessMode.NO_LOCK, 2000, "");
               ioDmm.IO.TerminationCharacterEnabled = true;

          }
          catch (SystemException ex)
          {
               MessageBox.Show("Open failed on " + this.txtAddress.Text + " " +
   ex.Source + "  " + ex.Message, "IdnSample", MessageBoxButtons.OK,
   MessageBoxIcon.Error);
               ioDmm.IO = null;
          }
      }
```

Switch operation

This section gives an overview of working with channels, including a discussion of channel types, selecting channels, opening and closing channels, setting common channel attributes, and setting up channel patterns.

NOTE

To install the switching card, refer to instructions in Series 3700A Quick Start Guide. For detailed information about the Series 3700A switch cards, refer to the Series 3700A Switch and Control Cards Reference Manual (Keithley part number 3700AS-909-01) on the Product Information CD-ROM that came with your instrument.

The switching channels of a Series 3700A have specific settings for switch-only operations and specific settings for switch with DMM operations. The Series 3700A accesses different settings based upon the close or open operation you specify. You can perform such operations on switching module channels, analog backplane relays, and channel patterns.

⚠ 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.

Identify installed switching cards

To identify installed switching cards from the front panel:

Press the **SLOT** key to scroll through the model numbers, descriptions, and firmware revisions of the installed switching cards.

To identify installed switching cards from the web interface:

1. Select the **Unit** page.
2. In the Report area, select the slots that you want information about.
3. Select **Firmware Revision**.
4. Click **Generate Report**. Information about the cards in the slots is displayed below the button.

To identify installed switching cards from the remote command interface:

Use `print(slot[X].idn)`to query and identify installed switching cards:

```
print(slot[X].idn)
```

Where: *X* = slot number (from 1 to 6)

Example

To get a list of all switching cards installed in the slots of a Series 3700A, send the following command over the remote command interface:

```
for x=1,6 do print (slot[x].idn) end
```

The response will be similar to the following:

```
3722, Dual 1x48 Multiplexer, 01.00a, <Module Serial Number>
3721, Dual 1x20 Multiplexer, 01.02a, <Module Serial Number>
Empty Slot
Empty Slot
Empty Slot
Empty Slot
```

Specifying a channel

The channels on the cards that you can use with the Series 3700A are referred to by a channel specifier. You will use the specifier to identify channels for use with close and open operations, scans, and channel patterns. The specifier is used for all interfaces (front panel, web, and remote command).

A channel specifier is a four or five-digit alphanumeric sequence. The first digit is always the slot number of the slot in which the card is installed in the instrument. The remaining digits vary depending on the type of card.

The following sections describe the channel specifier in more detail and provide generic examples (which may or may not be suitable for your installed cards).

Channel types

The channel types that are used to control relays include:

- Matrix
- Multiplexer (MUX)
- Backplane
- Digital I/O
- Totalizer
- Digital to analog converter (DAC)

The channel types that are available on a card are defined by the type of card. The documentation for your card model lists the available channel types.

Specify multiple channel numbers using lists and ranges (a sequence of channel numbers). Lists and ranges build on the individual channel specifier.

Matrix card channel specifiers

The channels on the matrix cards are referred to by their slot, bank, row, and column numbers:

- **Slot number**: The number of the slot in which the card is installed.
- **Bank number**: The bank number, if used by your card. See your card documentation.
- **Row number**: The row number is either 1 to 8 or A to Z. See your card documentation.

Column number: Always two digits. For columns greater than 99, use A, B, C and so on to represent 10, 11, 12, …; the resulting sequence is: 98, 99, A0, A1, …, A8, A9, B0, B1, …

Matrix channel examples

Specifier	Slot number	Bank number	Row number	Column number
1104	1	N/A	1	04
11104	1	1	1	04
1203	1	N/A	2	03
213A4	2	1	3	104
3112	3	N/A	1	12
62101	6	2	1	01

Analog backplane relay channel specifiers

The channels for slots with analog backplane relays are referred to by their slot, backplane, bank, and relay numbers:

- **Slot number:** The number of the slot.
- **Backplane number:** Always 9.
- **Bank number:** The bank number, if used by your card. See your card documentation for detail.
- **Analog backplane relay number**: The number of the backplane relay. Typically 1 to 6. See your card documentation for detail.

Backplane relay examples

Specifier	Slot number	Backplane number	Bank number	Backplane relay number
1914	1	9	1	4
1922	1	9	2	2
2924	2	9	2	4
3916	3	9	1	6

Multiplexer, digital I/O, totalizer, and DAC channel specifiers

The channels for multiplexer (MUX), digital I/O, totalizer, and digital to analog converter channels are referred to by their slot and channel numbers:

- **Slot number:** The number of the slot in which the card is installed.
- **Channel number:** The number of the channel (always three digits).

Specifier	Slot number	Channel number
1004	1	004
2050	2	050
3012	3	012
3003	3	003
2007	2	007
1020	1	020

Close and open channel operations and commands

Switching channels have specific settings for switch-only operations and specific settings for switch with DMM operations. For switch-only operation, there are three close methods and one open method. For switch with DMM operation, there is one close and one open method.

NOTE

You can use scans to perform a user-specified sequence of close and open operations on multiple channels for switch only applications or the switch with DMM applications. Refer to Scanning and triggering (on page 3-1) for information on scan operations.

The command or operation used to request the close or open specifies the completion of either a switch-only operation or a switch with DMM operation.

You can use the front panel **CLOSE** and **OPEN** keys to perform either switch only operations or switch with DMM operations on the selected channels. The operations of the keys depend on the DMM configuration attribute setting of the selected channel. Refer to Channel Attributes (on page 2-93) for more information on the DMM Configuration attribute.

- When the DMM configuration is set to "nofunction", the **CLOSE** and **OPEN** keys function as switch only operations in the same manner as `channel.close` and `channel.open` commands. When the DMM configuration is associated with a particular function (for example, DC Volts), the CLOSE and OPEN keys function as switch with DMM operations, that is, in the same manner as `dmm.close` and `dmm.open` commands.

NOTE

An error occurs if you attempt to perform a switch with DMM operation on an item that does not have an associated DMM function.

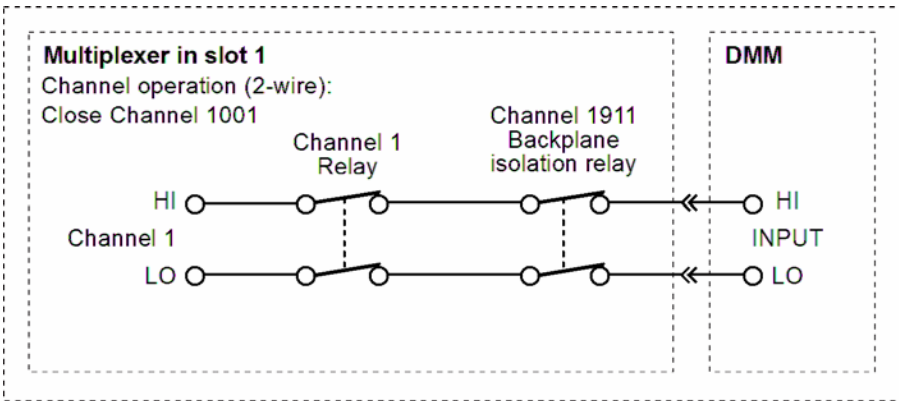
Corresponding remote commands for switch with DMM operations:

ICL command	Action performed
dmm.close()	Equivalent of channel.exclusiveslotclose except it also prepares the DMM for taking a measurement on the function associated with the item. It closes any needed backplane relays and paired channels. It opens channels and backplane relays that will interfere with measuring on the specified item.
dmm.open()	It opens the items that would get closed with a dmm.close().

When you perform a switch with DMM operation, the Series 3700A also closes the appropriate analog backplane relays to connect to the DMM Input and/or DMM Sense terminals. For 2-wire or two-pole DMM operations, the Series 3700A closes only the analog backplane relay to connect to the DMM Input terminal. For 4-wire or four-pole DMM operations, the Series 3700A closes the analog backplane relays to connect to the DMM Input and DMM Sense terminals.

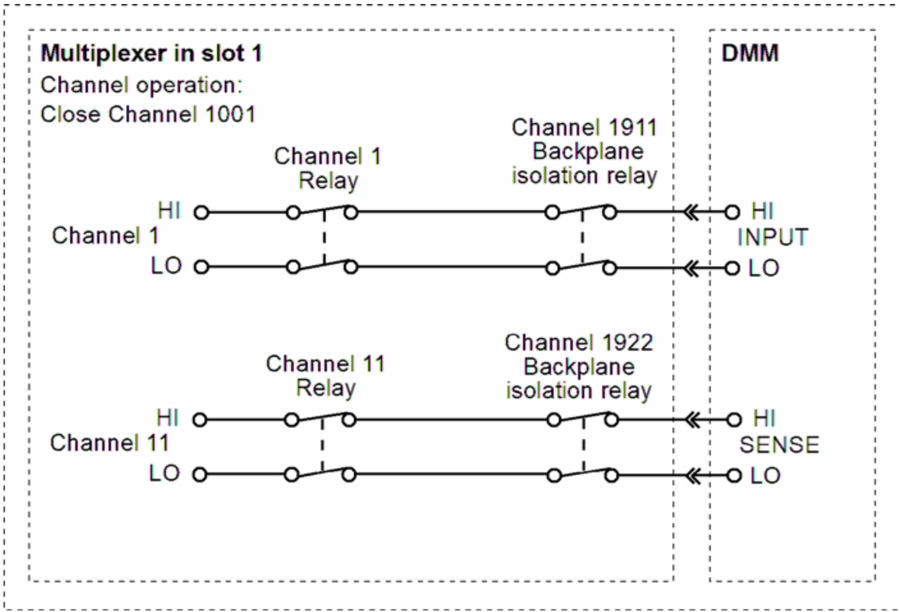
The following figure shows an example of how the channel is connected to the DMM Input of the Series 3700A for a 2-wire DMM operation. Assume a switching module with 20 channels is installed in Slot 1 of the mainframe and a 2-wire DMM operation, such as DC Volts, is selected. When you perform a DMM close operation on Channel 1001, the Series 3700A closes Channel 1001 and Channel 1911 (the backplane isolation relay) to connect the channel to the DMM Input terminal.

Figure 33: Two-wire function



The following figure shows an example of how the channel and its paired channel are connected to the DMM Input and Sense terminals of the Series 3700A for a 4-wire DMM operation. Assume a switching module with 20 channels is installed in Slot 1 of the mainframe, and a 4-wire function, such as 4Ω, is selected. When you perform a DMM close operation on Channel 1001, the Series 3700A closes Channel 1001 and Channel 1911 (backplane isolation relay) to connect the channel to DMM Input. The Series 3700A also closes Channel 1011 (the paired channel) and Channel 1922 (the sense backplane isolation relay) to connect the paired channel to DMM Sense.

Figure 34: Four-wire function



Selecting, closing, and opening channels

You can use the channel specifiers to select channels from the front panel, web interface, or over a remote command interface.

The methods for closing and opening channels include:

- Channel close: Close the selected channel
- Channel exclusive close: Close the selected channel and open any closed channels on the instrument (the only closed channel on the instrument is the one you selected)
- Channel exclusive slot close: Close the selected channel and open any closed channels in the same slot (the only closed channel on the slot is the one you selected)
- Channel open

The Series 3700A verifies that the operation being requested for a channel is supported by the specified channel and that the channel exists in the instrument.

NOTE

When you turn on the Series 3700A, relays for all switch cards in the instrument are opened. This includes all backplane relays.

Operating a channel from the front panel

CAUTION

Hot switching can dry-weld reed relays, causing them to always be on. Hot switching is recommended only when external protection is provided.

You can perform operations on a single channel from the front panel.

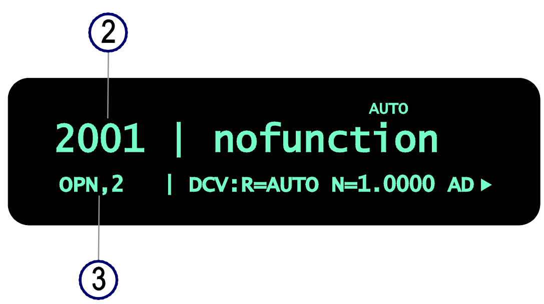
To select a channel:

1. If the instrument is being controlled remotely, press **EXIT** to allow control from the front panel.
2. Press the navigation wheel to select the first digit of the channel specifier, which is the slot number ①. The digit flashes, which indicates that it can be edited.

Figure 35: Select a channel from the Series 3700A front panel

3. To change to a different slot number, turn the navigation wheel until the slot number you want is displayed.
4. Press navigation wheel.
5. If your card supports banks, the next number you can select is the bank number. Set this as needed using the navigation wheel.
6. Set the channel number (or rows and columns for installed matrix cards) as needed using the navigation wheel ②.
7. The display shows the current state of the selected channel in the bottom row ③. In this example, the channel is open and 2-pole (if you see **:** followed by a channel specifier, you selected a range; press **EXIT** to return to the main display and reselect your channel).

Figure 36: Series 3700A selected channel state



8. To:
   - Close a channel without affecting any other channels: Select **CLOSE**.
   - Open the channel: Press **OPEN**.
   - Close a channel and open any other closed channels on the instrument: Select **CHAN** and select **EXCLOSE**. Press **ENTER** to close the selected channels.
   - Close a channel and open any other closed channels on the slot that contains the selected channel: Select **CHAN**, and then select **EXSLOTCLOSE**. Press **ENTER** to close the selected channels.

NOTE

Once a channel is selected, it is the selected channel for any subsequent front-panel operations.

Open and close channels from the Channel Action Menu

You can also use the options in the Channel Action Menu to open and close channels.

To use the Channel Action Menu to open and close channels:

1. Go to channel view.
2. Select the channel you want to open or close.
3. Press **CHAN**.
4. Use the navigation wheel to select the option. You can select:
   - OPEN: Opens the selected channel.
   - CLOSE: Closes the selected channel.
   - EXCLOSE: Closes the selected channel; opens any other channels that are closed.
   - EXSLOTCLOSE: Closes the selected channel; opens any other channels that are closed on the same slot.

1. Press the navigation wheel to open or close the channel.

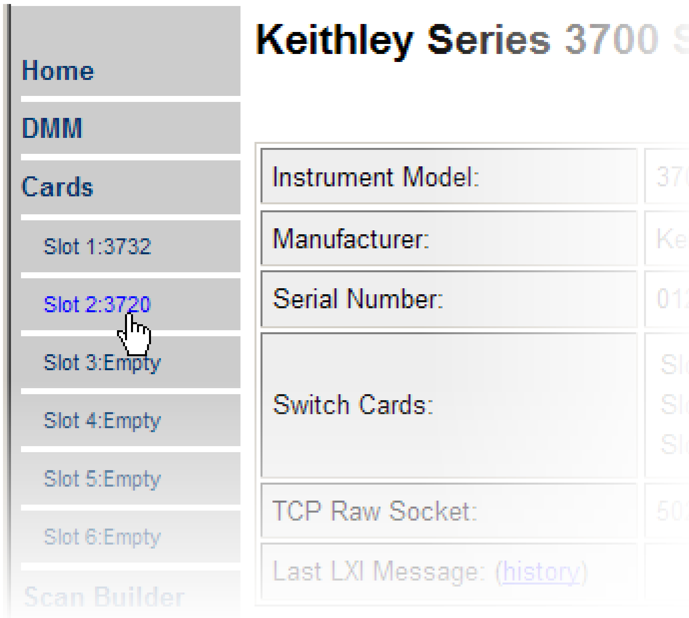
Selecting, closing, and opening a channel from the web interface

You can perform operations on a single channel from the web interface.

To select a channel:

1. You must log into the instrument to work with the channels. See Log in to the instrument (on page 2-36). After logging in, you can access the channel controls.
2. From the instrument home page, from the navigation on the left, select the slot that contains the channels you want to work with.

Figure 37: Web interface Cards list



3. To close a channel, click the channel. The display of the channel depends on the card that you have installed. Some examples are shown here.

Figure 38: Selecting, closing, and opening a channel from the web interface

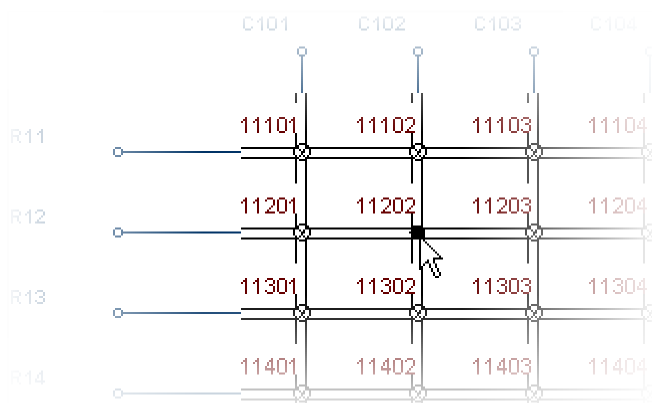


Figure 39: Selecting, closing, and opening a channel from the web interface

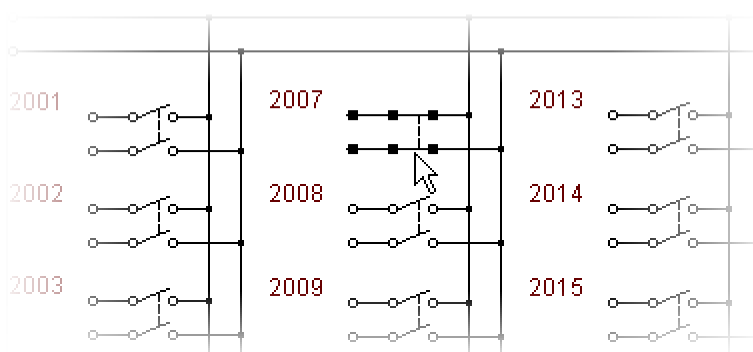
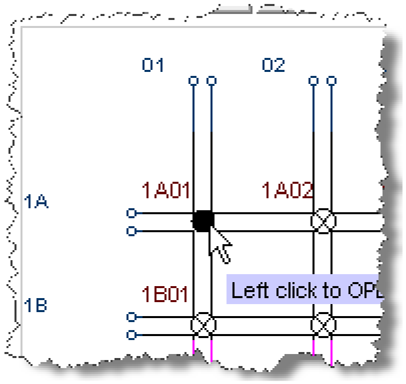


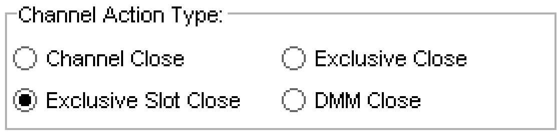
Figure 40: Close a channel



4. To open a closed channel, click it again.
5. To perform an exclusive close on a channel:

   Select **Exclusive Slot Close** in the Channel Action Type box. (Note that the DMM close option shown here is only available for instruments with the DMM feature installed.)

Figure 41: Selecting, closing, and opening a channel from the web interface



   a. Click a channel to close that channel and open all other channels.

Selecting, closing, and opening a channel using remote commands

To close or open a channel from the remote interface:

You can open and close channels using the following commands:

channel.close() (on page 8-50)
channel.exclusiveclose() (on page 8-56)
channel.exclusiveslotclose() (on page 8-57)
channel.open() (on page 8-79)

For example, to close channel 1001 over the remote interface, send the command:

```
channel.close("1001")
```

NOTE

Refer to the TSP commands (on page 8-10) for details on commands.

Channel list parameter for remote commands

The channel list parameter is a string-type parameter that is used when controlling the relays of the Series 3700A using a remote command interface. You can specify a list of individual channels or a range of channels in the channel list parameter.

In the command descriptions, the channel list parameter is shown as *`channelList`*.

When sending this parameter:

- Enclose the contents of the channel list in either single (') or double (") quotes. The beginning and end quotes must be the same style.
- Use a comma or semicolon to separate the channel list or channel patterns (on page 2-96).
- The string may contain a single channel, channel pattern, or analog backplane relay, as well as multiple ones that are indicated by a range or comma-delimited.
- Use a colon between the start and end channel to specify a range of channels. The lowest channel must be first and the highest last.

Examples:

- To perform an open or close operation on channels 1 and 3 of slot 1, use `("1001, 1003")` for the *`channelList`* parameter.
- To perform an open or close operation on all channels within the range of channels 1 through 5 of slot 1, use `("1001:1005")` for the *`channelList`* parameter

Queries that return a list of channels

For queries that return a channel list parameter, a channel configured for 4-pole operations will indicate the paired channel in parentheses. For example, if channel 3003 on a 60-channel card is configured for 4-pole, its paired channel is 3033. Notice the response to the query in the code example below:

```
channel.close('3003')
print(channel.getclose('slot3') → 3003(3033)
```

Return value

Several of the channel functions return a value for specified channels and channel patterns.

The return value for these functions is a string containing a list of comma-delimited return items. The *`channelList`* argument of the remote command determines the number and order of these returned items.

When the *`channelList`* parameter for these functions is `"slot`*`X`*`"`, the response first lists the channels starting from lowest to highest. More specifically, the channels are returned in numeric order.

When the *`channelList`* parameter for these functions is `"allslots"`, the response starts with slot 1 and increases to slot 6 for the Series 3700A. Each slot is processed completely before going to the next. Therefore, all slot 1 channels are listed before slot 2 channels.

When the response is numeric, but in string format, use the `tonumber()` function to convert the string to a number. For example, sending these commands:

```
x = tonumber("1403")
print(x)
```

Results in:

```
1.403000000e+03
```

When the response is a comma-delimited string, the individual return items can be identified by iterating through the list using the comma delimiters. For example, the Lua code below will start at the beginning of a string and break the string into individual items at each comma. The `tonumber()` function is used on each item to determine if it is a number or not. In either case, the value is printed.

```
index1 = 1
index2 = 1
text = "123,abc,hello,4.56"
endIndex = string.len(text)
while index2 ~= endIndex do
   index2 = string.find(text, ",", index1)
   if not index2 then
      index2 = endIndex
   end

   subString = string.sub(text, index1, index2 - 1)
   if not number(subString) then
      print(subString)
   else
      print(tonumber(subString))
   end
   index1 = index2 + 1
end
```

Selecting a range of channels on the front panel

You can perform operations on a single channel or range of channels. Specify a channel range by selecting a starting channel number and ending channel number. When you request an operation be performed on a range of channels, the Series 3700A performs the same operation on all channels within the channel range.

To select a channel range on the front panel (for example, channels 1003 through 1005):

NOTE

A single channel is selected when the starting and ending channel for a range match.

You cannot explicitly select an analog backplane relay on the front panel interface. You can only associate a backplane relay with a switching module channel. Refer to Channel attributes (on page 2-93) for further details.

1. To change the present slot, press the navigation wheel. The first digit of the four-digit channel number flashes, indicating edit mode.
2. Turn the navigation wheel to change the number to select any slot that has a switching module or pseudocard installed. For example, change the digit to a 1.
3. Press the navigation wheel a second time. This accepts the slot selection and selects edit mode for the channel. Digits two through four of the four-digit channel number flash, indicating edit mode.
4. Turn the navigation wheel to change the starting channel number You can select any channel available for the selected slot's module. For example, change the digits to 003.
5. Press the navigation wheel a third time. This accepts the channel selection and selects edit mode for the channel range. Digits two through four of the smaller four-digit channel number flash, indicating edit mode.
6. Turn the navigation wheel to change the ending channel number. You can select any channel available for the selected slot's module. For example, change the digits to 005.
7. Press the navigation wheel a fourth time to accept the channel selection.
8. Press the navigation wheel a fifth time to return to the main display after selecting the desired user configuration for the channel range.
9. Press the **CONFIG** key, followed by the **CHAN** key to change other channel attributes for the range. Similarly, press the **CHAN** key without the **CONFIG** key to bring up the CHANNEL ACTION MENU for use with the selected channel range.

Working with channels

Connection methods for close operations

You can dictate the order in which relays are opened and closed using the channel connection rule.

WARNING

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. This is the default behavior. When switch channels are both opened and closed, this command executes not less than the addition of both the open and close settle times of the indicated switch channels.

When the connection rule is set to 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.OFF`***), 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.***

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. This is the default.
- **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.
- **OFF**: Permits the instrument to initiate close and open operations simultaneously. This minimizes settling time for the close operation.

NOTE

You cannot guarantee the sequence of open and closure operations when the channel connect rule set to OFF. It is highly recommended that you implement cold switching when the channel connect rule is set to OFF.

To set the channel connect rule through the front panel interface:

1. Press the **MENU** key.
2. Use the navigation wheel to scroll to the CHANNEL menu item.
3. Press the **ENTER** key (or the navigation wheel) to display the CONNECT MENU.
4. From this menu, select the RULE menu item.
5. Set the rule to BBM, MBB, or OFF.
6. Use the **ENTER** key to apply the selection.
7. Use the **EXIT** key to leave the menu.

To set the channel connect rule through the web interface:

1. On the Unit page, in the upper left corner, select the channel connect rule menu.
2. Select Break Before Make, Make Before Break, or OFF.

To set the channel connect rule through the remote command interface:

Use the `channel.connectrule` command. Refer to the TSP commands (on page 8-10) for details.

Using sequential connect

During normal operation, the instrument attempts to minimize the duration of any channel action for a given card type and connect rule. This can result in multiple channels closing or opening simultaneously.

To prevent simultaneous closing and opening, you can use a sequential connection. A sequential connection ensures an orderly closing or opening of single individual channels in a channel list. An orderly action provides for:

- Repeatable and deterministic channel operation times
- Minimized power usage

You incur settling times at each close or open operation. If sequential connection is not selected, action settling times may vary depending on the card type. The total settling time is the sum of the settling times for each specified channel, plus any user delays that have been set for any closed channels. To better calculate timing, you can enable sequential channel connections. Deterministic implies that you can determine the time for a close operation to happen. For example, if you close three channels and each takes 4 ms to close, with sequential on, it will take 12 ms. With sequential off, it may be 4, 8 or 12 ms, depending on whether or not the card can close multiple channels at once.

Opening and closing relays in a sequential manner also uses minimum power. Since only one relay is closed or opened at any given time, the power used for that action is for a single relay and not additive.

By default, sequential connections are turned off. The order in which channels are opened or closed is not guaranteed. This feature also applies to scanning.

The sequential setting affects all channels in the instrument.

When specifying multiple channels for a single close or open operation, the total settling time depends on the relay drive scheme for the switching module — how each switching module budgets power to change the state of its relays. The Series 3700A supports the following relay drive schemes:

- Direct Drive: You can simultaneously update the state of all relays on a switching module with a single close or open operation. The total settling time for a close or open operation is the settling time for a single relay.
- Matrix Drive: You can execute a close or open operation on a list of channels, which can result in multiple actions to update the state of all specified relays. Settling time varies depending on the capabilities of the card and the number of relay closures.
- Hybrid Matrix Drive: For a single close or open operation, the state of all relays can be updated in no more than two steps. The total settling time for a close or open operation does is less than twice the settling time for a single relay.

To enable sequential connections through the front panel interface:

1. Press the **MENU** key.
2. Use the navigation wheel to scroll to the CHANNEL menu item.
3. Press the **ENTER** key.
4. Select the SEQUENTIAL menu item.
5. Select ON or OFF.
6. Use the **ENTER** key to apply the selection.
7. Use the **EXIT (LOCAL)** key to leave the menu.

To enable sequential connections through the web interface:

1. Open the **UNIT** page.
2. In the upper left corner, select the Sequential check box (next to the Channel Connect Rule list).

To enable sequential connections through the remote command interface:

Send the command:

`channel.connectsequential` (on page 8-53)

Determining the number of relay closures

The Series 3700A keeps an internal count of the number of times each switching card relay has been closed. The total number of relay closures is stored in nonvolatile memory on the switching card. Use this count to determine when relays require replacement (see the card documentation for information regarding the contact life specifications).

Relay closures are counted only when a relay transitions from open to closed state. If you send multiple close commands to the same channel without sending an open command, only the first closure is counted.

This option is not displayed if multiple channels are selected. A backplane relay has a close count associated with it as well; the number of closures are the closures that have occurred over the lifetime of the card.To view the close count for a channel from the front panel:

NOTE

You cannot query backplane relay closure counts through the front panel. You must use the remote command interface.

1. Use the navigation wheel ◷ to select the channel.
2. Press the **CONFIG** key.
3. Press the **CHAN** key.
4. Use the navigation wheel to scroll to the "COUNT" menu item.
5. Press the **ENTER** key (or the navigation wheel) to display the close counts.
6. Use the **EXIT** key to leave the menu.

To view the close count for a channel from the web interface:

1. From the list on the left, select a slot with an installed card.
2. Right-click a channel. The Channel Configuration dialog box is displayed.
3. Check the value in the Closure Count box.

NOTE

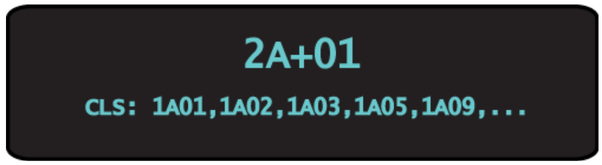
You can also work with channel patterns using the command `channel.getcount()`.

Viewing the close or open status of a channel

To determine whether a channel or backplane relay is closed or open, you can view its status using the front panel interface, remote command query, or instrument web page.

Viewing status from the front panel

Closed channels are shown on the display of the instrument, separated by commas. If more than one line of closed channels are displayed, you can press **DISPLAY** to display the full list. Use the navigation wheel to scroll through the list.



NOTE

For a four-pole operation the paired channel is not displayed on the front panel of the Series 3700A.

Viewing status from the remote command interface

To view a list of closed channels, use the `channel.getclose()` command. For example:

```
print(channel.getclose("allslots"))
```

To view the close and open status of channels, use the `channel.getstate()` command.

Viewing status from the instrument web page

To view status from the instrument web page, from the list on the left, select the slot that contains the channel. The status is displayed on the web page for the slot.

Channel attributes

You can use the front panel and command options to set attributes for specific channels. Some of the attributes you can set are adding a delay, forbidding closure of a channel, and setting channel labels, which are described in the following sections.

Setting and querying channel attributes

You can view and edit channel attributes on the front panel using the channel attributes menu. To access the channel attributes menu, press the **CONFIG** key and then press the **CHAN** key. Use the navigation wheel and **CURSOR** keys to change attribute values. Use the **ENTER** and **EXIT** keys to apply or cancel settings.

With the exception of the DMM configuration attribute, you can view and edit channel attributes using the remote command interface using the commands residing in the channel logical instrument. For example, to set the label attribute of a channel, use the `channel.setlabel` command; to retrieve the label attribute of a channel, use the `channel.getlabel` query.

To set the DMM configuration attribute for a channel or group of channels, use the `dmm.setconfig` command and specify the desired channels in the `<ch_list>` parameter. To retrieve the DMM configuration attribute for a channel or group of channels, use the `dmm.getconfig` query.

For specific instructions on retrieving the relay closure count attribute, refer to Relay closure count (see "Determining the number of relay closures" on page 2-92).

Set additional delay

You can set an additional delay to incur after the relay settles when closing.

To set additional delay time from the front panel:

1. Display a channel (you might need to press **DISPLAY**).
2. Select the channel for which you want to set attributes.
3. Press **CONFIG**, then press **CHAN**.
   - **DELAY:** Additional delay to incur after the relay settles. Enter the value for the delay in seconds. The total delay for channel operation is user delay plus the relay settling time.

To set additional delay time from the web interface:

1. From the list on the left, select the slot that contains the channel you want to set an additional delay on.
2. Right-click on the channel you want to bring up the channel configuration dialog box for that channel.
3. Enter the desired delay time (in seconds) in the delay time field on the right side of the dialog box. Once the desired time is entered, click **OK**.

To set additional delay time through the remote interface:

Use the command:

`channel.setdelay()` (on page 8-93)

Forbid closing a channel

You can prevent a channel from being closed from any interface by setting it to forbidden.

> **NOTE**
>
> If the channel that is to be forbidden is used in a channel pattern, the pattern is deleted when you set the channel to be forbidden to close. An analog backplane relay can be marked as forbidden to close. Analog backplane relays only support the forbidden setting attribute.

To forbid closing of a channel from the front panel:

1. Display a channel (you might need to press **DISPLAY** first).
2. Select the channel for which you want to set attributes.
3. Press **CONFIG**, then press **CHAN**.
4. Use the navigation wheel ◷ to select **FORBID**.
5. Select Yes to prevent a channel from being closed or No to allow closures.
6. Press the navigation wheel ◷ to save the change.

To forbid closing of a channel from the web interface:

1. From the list on the left, select the slot that contains the channel you want to forbid close on.
2. Right-click the channel.
3. Select the forbidden checkbox.
4. Click **OK**.

To forbid closing of a channel from the remote interface:

You can also set this attribute using the following commands:

- `channel.setforbidden()` (on page 8-94)
- `channel.clearforbidden()` (on page 8-49)

Set up labels

You can define labels for channels.

Labels must be unique; they cannot have the same as the name of another channel or channel pattern. Labels cannot contain spaces, and they do not persist through a power cycle.

You cannot apply a label to a range of channels.

Channel labels can be up to 19 characters.

You can only set labels for channels that are installed in the instrument.

To set up labels from the front panel:

1. Display a channel (you might need to press **DISPLAY** first).
2. Select the channel for which you want to set labels.
3. Press **CONFIG**, then press **CHAN**.
4. Use the navigation wheel ◷ to select **LABEL**, which allows you to set the label that will show on the front-panel for the specified channel.
5. Change the name using the navigation wheel ◷.
6. Press the navigation wheel ◷ to save the change.

To set up labels from the web interface:

1. From the list on the left, select the slot that contains the channel you want to set up a label on.
2. Right-click the channel.
3. In the Label box, enter the label.
4. Click **OK**.

To set up labels from the remote interface use the channel.setlabel() (on page 8-94) command.

You can use labels to refer to the channels in commands. For example, if you set the label for channel 3005 to "start", you could use "start" to close and open the channel.

This is shown in the following example:

```
channel.setlabel("3005","start")
channel.close("start")
print(channel.getclose("allslots"))
```

Pole settings

- **BACKPLANE RELAYS:** List of backplane relays to control when performing a switch-only operation on a single channel or range of channels. This attribute is not applicable to channel patterns. Refer to Channel patterns.
- **POLE SETTING:** Pole setting for multiplexer (MUX) channels indicates if the paired MUX channel should be included when performing a close or open operation on channel.
  In a switching module that has 60 channels, the Series 3700A automatically pairs Channels 1 through 30 with Channels 31 through 60 (respectively) when the pole setting for a channel is set to 4-pole. Once you configure the pole setting of a switching channel for 4-pole, the associated paired channel becomes unavailable for switching operation. For example, assume 3003 is set to 4-pole and its paired channel is 3033. Now, you cannot set attributes or perform close/open operations on 3033. A paired channel settings conflict error generates if you specify Channel 3033 for a close/open operation.

NOTE

Matrix channels have fixed pole settings. Multiplexer channels pole settings may be changed.

Channel patterns

You can use channel patterns as a convenient way to refer to a group of switching channels and backplane relays with a single alphanumeric name. When you perform close or open operations on a channel pattern, only the channels and analog backplane relays that are in the channel pattern are affected.

There is no speed difference when performing close and open operations on channel patterns compared to performing the same operations on individual channels or a list of channels.

Assigning channel pattern attributes

A channel pattern has only two attributes: the channel pattern name and a DMM configuration. An error occurs if you attempt to assign or query any channel attributes other than DMM configuration for a channel pattern.

You associate a name with a channel pattern when you create the pattern.

To assign a DMM configuration to a channel pattern using the front panel interface use the PATTERN ATTRibutes menu. You must create the channel pattern before you can access the PATTERN ATTRibutes menu. To access this menu after creating a channel pattern, press the **CONFIG** key followed by the **PATT** key.

To assign a DMM configuration to a channel pattern using the remote command interface, use the `dmm.setconfig` command and specify the channel pattern name for the *`channelList`* parameter. To retrieve the DMM configuration attribute for a channel pattern, use the `dmm.getconfig` query and specify the channel pattern name for the *`channelList`* parameter.

Pole settings and channel patterns

NOTE

Changing a channel's pole setting deletes all patterns containing that channel.

Set the pole setting of switching module channels prior to creating a channel pattern image. If you change the pole setting for a channel, the Series 3700A will delete any patterns that contain that channel. For example, assume a channel pattern called 'myimage' has channels 2004, 2008 and 2012 associated with it while 'myimage2' has channels 2005, 2009 and 2011. Now, if pole setting of Channel 2004 changes then the channel pattern 'myimage' is deleted and no longer exists in system. However, the pattern called 'myimage2' still exists.

While creating channel pattern images, the paired channel is automatically accounted for based on pole setting. Therefore, you do not need to manually specify the paired channel in the channel pattern image. For example, assume Slot 1 has a 3720 card installed and all channels are set to 4-pole operation. With all channels configured for 4-pole, the available channels are 1001 to 1030. To create a channel pattern called 'one4wire' with Channel 1001 and backplane relays 1911 and 1922, the corresponding bus command is:

```
channel.pattern.setimage('1001, 1911, 1922', 'one4wire')
```

To see the image associated with a channel pattern, use the `channel.pattern.getimage` command. For example, to see the image of the pattern, just created called 'one4wire':

```
print(channel.pattern.getimage('one4wire')) → 1001(1031),1911,1922
```

NOTE

Paired channel are indicated in parentheses in <ch_list> queries.

Create a channel pattern

When you create a channel pattern, make sure to:

- Include all the channels and backplane relays that are needed for that channel pattern.
- Check that channels contained in the pattern are correct.
- Check that channels contained in the pattern create the desired path connection.
- Make sure that channels that you want to include in the pattern are not set to forbidden to close.

When naming the channel pattern, be aware:

- The first character of the name must be alphabetic (upper or lower case)
- Names are case sensitive
- Pattern names must be different than channel labels

Performing close and open operations on channel patterns

WARNING

Careless channel pattern operation could create an electric shock hazard that could result in severe injury or death. Improper operation can also cause damage to the switching cards and external circuitry. The control of multiple channels using channel patterns should be restricted to experienced test engineers who recognize the dangers associated with multiple channel closures.

You can close and open channel patterns the same way you do for individual channels.

When you request a close or open operation, the Series 3700A verifies that the channels exist for a pattern, but does not verify that the switch path connection is correct. You must ensure the requested operation is safe for a channel pattern and that a good connection will result for your application with the channel pattern.

Channel pattern storage

Channel patterns are:

- Part of saved setup data and restored when a setup is recalled.
- Deleted when the instrument is reset or has a pole setting change.
- Deleted when a channel associated with the pattern is reset.
- Allocated 32KB of memory in the Series 3700A instrument for all channel patterns.

The number of channel patterns you can store varies with the number of characters of the channel pattern name, the number of characters used in listing the switching channels, and the number of characters in the name of the DMM configuration. 32KB of memory is equivalent to 32,000 characters. If each channel pattern name is five characters long, and each pattern is comprised of five channels, and the channel list is comma delimited (for example, "2003,4003,2005,4005,2915"), then you can store 642 channel patterns. You can store additional channel patterns by decreasing the number of characters in each channel pattern name or the number of channels in the channel pattern image. Conversely, you store fewer than 642 channel patterns by increasing the number of characters in the channel pattern name or number of channels in the channel pattern image.

To see how much of the channel pattern memory is available or used, send the command:

```
print(memory.available())
```

or

```
print(memory.used())
```

Refer to `memory.available()` (on page 8-302) or `memory.used()` (on page 8-303).

Reset a channel

You can reset a channel to its factory default settings. When you reset a channel:

- Any closed channels and analog backplane relays open
- The poles of all channels reset to 2-pole operation and paired channels are changed to match
- Additional user delay is set to zero
- Labels return to default of SCCC or slot, row, column
- Analog backplane relays specified by the `channel.setbackplane()`function are cleared
- If the channel is forbidden to close, it is cleared from being forbidden to close
- If the channels are used in channel patterns, the channel patterns that contain the channels are deleted.
- The DMM configurations of all channels are set to `nofunction`

Using this function to reset a channel or backplane relay involved in scanning invalidates the existing scan list. The list has to be recreated before scanning again.

CAUTION

Resetting a channel deletes any channel patterns that contain that channel.

To reset a channel from the front panel:

1. Display a channel.
2. Select the channel you want to reset.
3. Press **CHAN**.
4. Select **RESET**.
5. Select SELECTED, ALL, or CANCEL.
6. Press the navigation wheel to reset the channel.

To reset all channels on a slot from the web interface:

1. Select the slot that contains channels you want to reset.
2. Click **RESET SLOT**.
3. All channels on the slot are reset.

To reset a channel from the remote interface:

Send the command `channel.reset()` (on page 8-87).

Pseudocards

You can perform open, close, and scan operations and configure your system without having an actual switching card installed in your instrument. Using the remote interface, you can assign a pseudocard to an empty switching card slot, allowing the instrument to operate as if a switching card were installed.

A pseudocard cannot be configured from the front panel. However, once the remote configuration is complete, you can take the instrument out of remote mode and use the front panel. Press the **EXIT** key to take the instrument out of remote mode.

When the instrument is turned off, the pseudocard is no longer assigned to the slot.

NOTE

A saved setup or created configuration script retains the model number of the card installed in each slot. The model number of a pseudocard is the same as the model number of an actual card (except for Model 3732 cards; see the "Pseudocard support for the Model 3732" topic in the Series 3700 Switch and Control Cards Reference Manual for details). This allows a saved setup or created configuration script to be recalled if the installed card (or pseudocard) matches the model number for the slot in the saved setup or created configuration script.

Pseudocards programming example

Use the following command to set the pseudocard of slot 6 for 3720 Dual 1 x 30 Multiplexer card simulation:

```
slot[6].pseudocard = 3720
```

Alternatively, you could send the following command:

```
slot[6].pseudocard = 3720
```

Save the present configuration

You can capture the present settings of the instrument using the create configuration script feature. When you run this feature, the configuration script is created and saved. You can run it later to return to that configuration, or set it up to be the autoexec script. The configuration script is a normal TSP script; once created, you can use it and modify it as you would any other script.

The configuration script includes:

- Comment lines that identify the script as auto created and the date and time of creation.
- The cards that are installed and the slots in which they are installed.
- A reset command, which will reset the instrument to the factory default settings.
- The commands to reconfigure the instrument. The configuration script only captures settings that have been changed from the factory defaults.

Later, when you run the configuration script, the script will verify that the installed cards and slots match. If they do not, a message is displayed, the script stops, and the configuration is not restored.

Note that the configuration script does not include the status of channels. As initially created, the configuration script performs a reset, which opens all channels.

NOTE

You can modify the script to change the card models or slots. However, you must make sure that all subsequent commands are valid for the card model or slot change.

NOTE

For more information on scripts, see Fundamentals of scripting for TSP (on page 7-1). For more information on the autoexec script, see Autoexec script (on page 7-7).

A sample configuration script is shown in the following example.

`--Auto created configuration script`	Indicates that this was created with the Create Configuration Script feature
`--Tue Jul 13 13:02:12 2010`	Date and time stamp
`if string.find(slot[1].idn, "7174") == nil then` `   print(` `      "Card installed in slot 1 needs to be a 7174.")` `   display.clear()` `   display.settext("Card installed in$N" ..` `      "$Bslot 1$R needs to be a $B7174$R")`	Code that verifies that card and slot are in agreement
`else`	
`   reset()`	Reset command
`   channel.setlabel("1A01", "FirstRowCol")` `   channel.setlabel("1A12", "LastRowCol")` `   channel.setlabel("1B01", "FirstNextRow")` `   channel.setlabel("1B12", "LastNextRow")` `   channel.pattern.setimage("1A01,1B01","Row1_2_col_1")` `   channel.pattern.setimage("1A02,1B02","Row1_2_col_2")` `   channel.pattern.setimage("1A03,1B03","Row1_2_col_3")` `   channel.pattern.setimage("1A04,1B04","Row1_2_col_4")` `   channel.pattern.setimage("1A05,1B05","Row1_2_col_5")` `   channel.pattern.setimage("1A06,1B06","Row1_2_col_6")` `   channel.pattern.setimage("1A07,1B07","Row1_2_col_7")` `   channel.pattern.setimage("1A08,1B08","Row1_2_col_8")` `   channel.pattern.setimage("1A09,1B09","Row1_2_col_9")` `   channel.pattern.setimage("1A10,1B10","Row1_2_col_10")` `   channel.pattern.setimage("1A11,1B11","Row1_2_col_11")` `   channel.pattern.setimage("1A12,1B12","Row1_2_col_12")` `      collectgarbage()` `   scan.trigger.channel.stimulus =` `      scan.trigger.EVENT_CHANNEL_READY` `   scan.create()` `   scan.mode = 0` `   scan.bypass = 1` `   scan.add("Row1_2_col_1")` `   scan.add("Row1_2_col_2")` `   scan.add("Row1_2_col_3")` `   scan.add("Row1_2_col_4")` `   scan.add("Row1_2_col_5")` `   scan.add("Row1_2_col_6")` `   scan.add("Row1_2_col_7")` `   scan.add("Row1_2_col_8")` `   scan.add("Row1_2_col_9")` `   scan.add("Row1_2_col_10")` `   scan.add("Row1_2_col_11")` `   scan.add("Row1_2_col_12")`	Code that captures the non-factory default settings
`end`	

Create a configuration script

When you run the create configuration script feature, it automatically generates a user script that is saved to a script with a name that you define. Create configuration script is available from the front panel of the instrument, the web interface, and the remote interface.

NOTE

When you specify the name of the script, be aware that if you specify a name that already exists (including `autoexec`), the existing script is overwritten with the new configuration script.

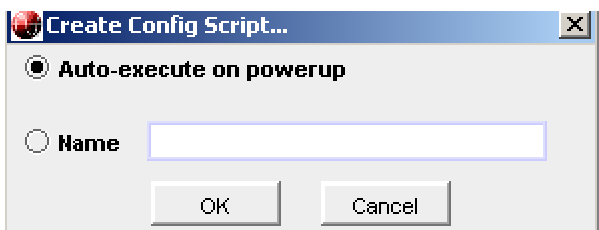
To create a configuration script from the front panel:

1. Press **MENU**.
2. Select **SCRIPT**.
3. Select **CREATE-CONFIG**. The AUTOEXEC ON PWR UP prompt is displayed.
4. Select **Yes** or **No**.
5. If AUTOEXEC is set to no, at the name prompt, enter the name of the configuration. The default name is config01.
6. Press **ENTER**.
7. The AUTOEXEC message is displayed again. Press **EXIT** several times to return to the normal display.

To create a configuration script from the web interface:

1. Open the **Unit** page.
2. Log in if necessary.
3. Click **Create Config Script**.

Figure 42: Create Config Script dialog box



4. To make the configuration script the autoexec script, select Auto-execute on powerup.
5. To assign a name (the script will not be the autoexec script), select Name and enter a name in the box.
6. Click **OK**. The configuration script is created.

To create a configuration script from the remote interface:

Send the command:

```
createconfigscript(name)
```

Where *name* is the name you want to assign to the configuration script.

Running the configuration script

You can run the configuration using the same methods as any other script. See Run scripts (on page 7-5) for information.

Section 3

Functions and features

In this section:

Scanning and triggering

Introduction to scanning and triggering

A scan is a series of steps that opens and closes switches sequentially for a selected group of channels. During each step, actions occur, such as waiting for a trigger, taking a measurement, and completing a step count. Scans automate actions that you want to perform consistently and repeatedly on a set of channels.

Triggers are events that prompt the instrument to move from one step to another in a scan. Triggers can come from a variety of sources, such as a key press, digital input, or expiration of a timer. The sequence of actions and events that occur during the scan is called the trigger model.

Scanning and triggering allow you to synchronize actions across channels. You can set up a scan using the trigger model to precisely time and synchronize the Series 3700A between channels and multiple instruments. You can also use triggers without the triggering model to set up a scan to meet the needs of a specific application that does not fit the triggering model.

You can configure and run scans from the front panel, over a remote communication interface, or through the web interface. If you are using the communication interface or the web interface, the scan is a set of actions determined by the trigger model. If you use the front panel, key presses determine the order of the scan.

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

NOTE

If desired, readings for scanned channels may be automatically stored in a specified reading buffer (Buffer: Data Storage and Retrieval).

Trigger model

When you run a scan, the scan sequence follows a trigger model. The trigger model is shown in the following flowchart.

NOTE

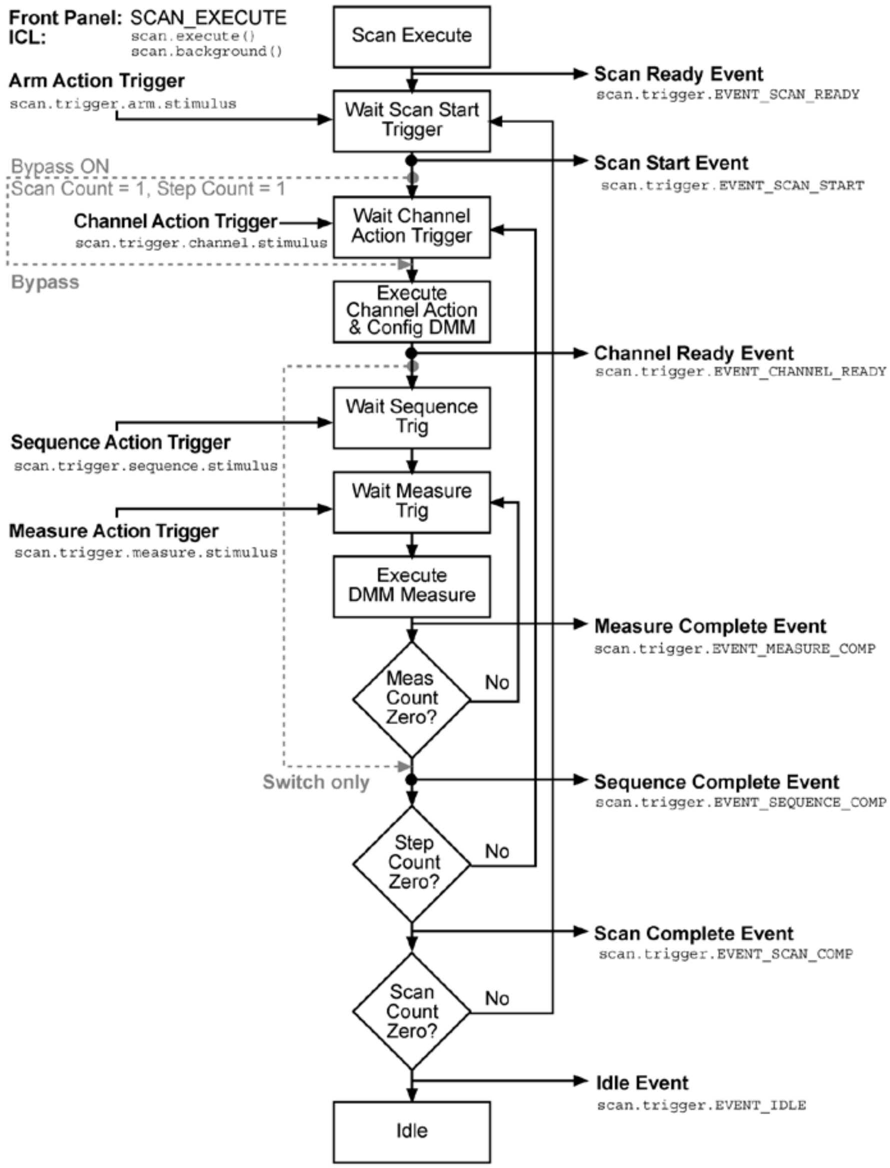
In Series 3700A, only scanning operations use the trigger model. Individual open, close, and measure commands do not affect the trigger model.

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

NOTE

You cannot use an external trigger event (for example, digital I/O) for the channel stimulus setting of the trigger model when using the front-panel STEP key.

Figure 43: Trigger model



Trigger model components

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

Trigger model events and associated commands

The Model 3706A trigger model has the following events and associated command attributes. These events, along with other events in the system, may be used to configure various stimulus settings.

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

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

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

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

Event	Associated attribute
Scan Ready Event	`scan.trigger.EVENT_SCAN_READY`
Scan Start Event	`scan.trigger.EVENT_SCAN_START`
Channel Ready Event	`scan.trigger.EVENT_CHANNEL_READY`
Measure Complete Event	`scan.trigger.EVENT_MEASURE_COMP`
Sequence Complete Event	`scan.trigger.EVENT_SEQUENCE_COMP`
Scan Complete Event	`scan.trigger.EVENT_SCAN_COMP`
Idle Event	`scan.trigger.EVENT_IDLE`

NOTE

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

Scan and step counts

When running a scan, it may be necessary to determine the scan progress. You can use `scan.state()` to read the scan and step count to determine the point in the scan table being executed.

"Scan count" represents the number of the current iteration through the scan portion of the trigger model. This number does not increment until after the scan begins. Therefore, if an 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 yet to begin, the scan count is zero.

"Step count" represents the number of times the scan has completed a pass through the channel action portion of the trigger model. This number does not increment until after the action completes. Therefore, if the instrument is waiting for an input to trigger a channel action, the step count represents the previous step. If no step has yet to complete, 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.

For example:

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

Basic scan procedure

NOTE

It is always better to configure all channel and DMM attributes before creating a scan.
You cannot use an external trigger event, like digital I/O, for the channel stimulus setting of the trigger model when using the front panel **STEP** key. For more information, see Scanning and Trigger model (on page 3-2).

To perform a scan:

1. Configure the channels for scanning as needed.
2. Select (or create, if necessary) the reading buffer to store measurements (if desired).
3. Build the scan list:
   - **Front panel**: Press the INSERT key. The steps are executed in the order in which they are added. When adding a range of channels, they are added to the end of the existing scan list.
   - **Remote interface:** Send the `scan.create()`, `scan.add()` , or `scan.addimagestep()` command.
4. Configure the scan settings (for example, scan count, bypass, mode, and so on).
5. To start the scan:
   - **Front panel**: Press the **STEP** key or the **SCAN** key and select the **BACKGROUND** menu item.
   - **Remote interface**: Send the command `scan.execute` or `scan.background`.
6. The trigger model leaves the idle state and performs actions on the channels involved in scanning, along with channels that would interfere with scanning, such as AMP channels, analog backplane relays 1 and 2 on all slots, commonside ohm backplane channels, and other channels in banks involved in scanning.
   - **Front panel:** When you press the **STEP** key, the Model 3706A leave the idle state and perform the channel action associated with the first step in the scan list.

   Measurements are then taken (if part of the scan). If a reading buffer was selected, the result from the measurements are stored there. The measurement action, if started, is completed. The channel and DMM remain as previously configured until the next step in the scan is initiated. The DMM configuration changes to the attribute settings tied to the channel in the next step.

NOTE

While scanning is enabled, pressing most front panel keys will cause the instrument to display error code 5522, "Scan Running, Must Abort Scan."

1. The channels are scanned or stepped in the order they were added to the list.
   - **Front panel**: If you are stepping through the scan, press the **STEP** key to proceed to the next step in the list.
   - **Remote interface**: You cannot step a scan remotely over the bus.
2. To abort the scan:
   - **Front panel**: Press the **EXIT** key.
   - **Remote interface**: Use the `scan.abort()` ICL command.

NOTE

Even if the scan is aborted, the DMM remains as configured in the last completed step of a scan that involved measuring and channel states match the aborted state of channels in terms of which are closed and opened.

The DMM remains as previously configured in the last completed measurement step of a scan that involved measuring. The function associated with that configuration will have the associated DMM attributes updated to match. All other functions will remain as configured prior to scanning.

If configured to scan the channels in the scan list again, the Series 3700A waits at the control source for another trigger event. After the scan is complete, the Series 3700A outputs another trigger pulse, if configured to do so. After all requested scans are complete, the instrument returns to the idle state with the channels associated with last scan step closed.

Buffer

To recall scanned readings stored in the buffer, press the **REC** key and turn the navigation wheel to navigate through the buffer. See Recalling readings (on page 3-52) for details on recalling buffer readings. When finished, make sure to exit from buffer recall (press the **EXIT** key). Also see Reading buffers (on page 3-55, on page 3-49).

Changing attributes of an existing scan

When a scan already exists, changing channel and DMM attributes also causes the scan to change. Once a scan list has been defined, the Series 3700A tries to incorporate your changes into the scan. For example, changing a DMM configuration assigned to a channel used in scanning affects the scan list. But changing a DMM configuration on a channel not involved in scanning does not affect the scan list. If the change impacts the ability of the scan to function properly (such as deleting something referenced by the scan), an error message is logged and the scan list may be cleared.

To see how the scan list may have changed, view the current scan list:

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

For remote operation, use the `scan.list()` function.

For performance reasons, it is always better to configure all channel and DMM attributes before creating a scan. Afterward, changes may cause the scan to take more time to modify the scan list.

You can clear an existing scan list before making any changes after making a scan list. From the front panel, press the **SCAN** key and select the **CLEAR** option. For bus operation, use the `scan.create()` function.

Some changes may cause channels to be dropped from the list when they become paired with another channel for a 4-wire operation. These channels will not be added back into the list during subsequent changes that free the paired channel from a 4-wire operation. To get a recently unpaired channel back in the list, create a new scan list or add it back into the list.

For example, a scan list is comprised of Channels 1 to 60 on a Model 3720 card with the channels configured to measure DC volts. Changing Channels 1 to 30 to be configured for 4-wire ohms measurements causes the scan list to change. The scan list changes to contain Channels 1 to 30 measuring 4-wire ohms, and Channels 31 to 60 are removed because they are paired with Channels 1 to 30. If you then change Channels 1 to 60 to be configured for measuring DC volts, the scan list will still only contain Channels 1 to 30, but it will be measuring DC volts. Channels 31 to 60 are not automatically added back into the list.

The remote commands to simulate this example follow. Assume the Model 3720 is in Slot 3:

```
-- Configure Channels 1 to 60 to measure DC volts.
dmm.setconfig("slot3", "dcvolts")
-- Create a scan list, channels measuring DC volts.
scan.create("slot3")
-- View the scan list, 60 channels measuring DC volts.
print(scan.list())
-- Change Channels 1 to 30 to 4-wire ohms.
dmm.setconfig("slot3", "fourwireohms")
-- List now has Channels 1 to 30 measuring 4-wire ohms.
print(scan.list())
-- Change back to DC volts on Channels 1 to 60.
dmm.setconfig("slot3", "dcvolts")
-- List still has Channels 1 to 30, but measures DC volts.
print(scan.list())
```

To configure a scan from the SCAN ATTR MENU, while in an active scan list:

1. Press the **CONFIG** key.
2. Press the **SCAN** key. Modify any of the following menu items as desired:
   - **ADD**: Displays **Use INSERT** key. The related command is `scan.add`, without the optional DMM configuration.
   - **BYPASS**: Enables (ON) or disables (OFF) bypassing the first step of the first scan pass. Related command: scan.bypass (on page 8-324).
   - **MODE**: Sets the scan mode value to one of the following:
     - OPEN_ALL (default setting)
     - OPEN_SELECT
     - FIXED_ABR

     Related command: scan.mode() (see "scan.mode" on page 8-331).
   - **MEAS_CNT**: Sets the measure count value. Related command: scan.measurecount (on page 8-330).
   - **SCAN_CNT**: Sets the scan count value. Related command: scan.scancount (on page 8-335).
3. Press the **EXIT** key to leave the menu.

Front-panel scanning

After channels have been added to the scan list, press the **SCAN** key to display the SCAN ACTION MENU. If no scan list exists, pressing the **SCAN** key will briefly display "No Scan List. Use INSERT to add selection."

The menu contains the following items:

- **BACKGROUND:** Runs scan list in the background
- **CREATE**: Displays **Use INSERT key**
- **LIST:** Displays the current scan list steps. Turn the navigation wheel to scroll through the list.
- **CLEAR:** Clears the existing scan list.
- **RESET:** Resets the unit's scan settings, which include scan count, clearing the scan list, and scan stimulus settings like scan trigger arm.

Press the **INSERT** key to add the selected channels or pattern to the existing scan list.

Press the **DELETE** key to remove the selected channels or pattern from the existing scan list. Only the first occurrence of the selected item is removed. For example, if Channel 3003 appears in the list three times and Channel 3003 is selected when the **DELETE** key is pushed, the first step using Channel 3003 will be removed (the remaining two will stay in the list).

When removing channels, channel patterns are not checked to determine if the channel being removed is associated with its image. To remove a channel pattern in a scan list, select the channel pattern to be removed, and then press the **DELETE** key. Continuing the previous example of Channel 3003, if 'mypat1' is comprised of Channels '3003, 3033, 3911, and 3922' when the remove request for Channel 3003 is made, it will not remove 'mypat1' from the list. To remove 'mypat1' from list, select the channel pattern 'mypat1' and press the **DELETE** key, which removes the step and all associated channels.

Press the **STEP** key to single step through a scan list.

Foreground and background scan execution

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

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

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

Most of the commands to change how the instrument is configured will log the following error message to the error queue:

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

Include multiple channels in a single scan step

Through the remote control interface, you can use `scan.addimagestep` to combine a list of channels into a scan step.

The following example creates five scan steps with the indicated channels.

```
scan.create()
scan.addimagestep("1A01, 1B01, 1C03")
scan.sddimagestep("1A03, 1B03, 1C03")
scan.addimagestep("1A05, 1B05, 1C03")
scan.sddimagestep("1A07, 1B07, 1C03")
scan.addimagestep("1A09, 1B09, 1C03")
```

Remote interface scanning

Scan and trigger commands

The following list contains commands associated with triggers and bus operation scanning:

- lan.trigger[N].clear() (on page 8-282)
- trigger.blender[N].stimulus[M] (on page 8-425)
- trigger.blender[N].wait() (on page 8-427)
- trigger.timer[N].clear() (on page 8-429)
- trigger.timer[N].stimulus (on page 8-433)
- digio.trigger[N].clear() (on page 8-122)
- digio.trigger[N].pulsewidth (on page 8-125)
- digio.trigger[N].stimulus (on page 8-127)
- digio.trigger[N].wait() (on page 8-129)
- lan.trigger[N].assert() (on page 8-281)
- lan.trigger[N].clear() (on page 8-282)
- lan.trigger[N].overrun (on page 8-287)
- lan.trigger[N].stimulus (on page 8-288)
- lan.trigger[N].wait() (on page 8-291)
- scan.add() (on page 8-319)
- scan.background() (on page 8-323)
- scan.bypass (on page 8-324)
- scan.create() (on page 8-325)
- scan.execute() (on page 8-327)
- scan.list() (on page 8-328)

Hardware trigger modes

Use the hardware trigger modes to integrate Keithley Instruments and non-Keithley instruments into an efficient test system. The hardware synchronization lines are classic trigger lines. The Series 3700A contains 14 digital I/O lines and three TSP-Link synchronization lines that you can use for input or output triggering. The following table provides a summary for each hardware trigger mode.

Trigger mode	Output		Input	Notes
	Unasserted	**Asserted**	**Detects**	
Bypass	N/A	N/A	N/A	Use the `writebit` and `writeport` commands for direct line control
Either edge	High	Low	Either	Short input pulses can cause a trigger overrun
Falling edge	High	Low	Falling	
Rising edge	N/A	N/A	N/A	• The programmed state of the line determines if the behavior is similar to RisingA or RisingM • High similar to RisingA • Low similar to RisingM
Rising A	High	Low	Rising	

Trigger mode	Output		Input	Notes
	Unasserted	**Asserted**	**Detects**	
RisingM	Low	High	None	
Synchronous	High latching	Low	Falling	• Behaves similar to SynchronousA • Trigger overrun detection is disabled • To mirror the SynchronousA trigger mode, set the pulse duration to 1 µs or any small nonzero value
SynchronousA	High latching	High	Falling	Ignores the pulse duration
SynchronousM	High	Low	Rising	

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

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

For additional information on the hardware trigger modes, see TSP commands (on page 8-10).

NOTE

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

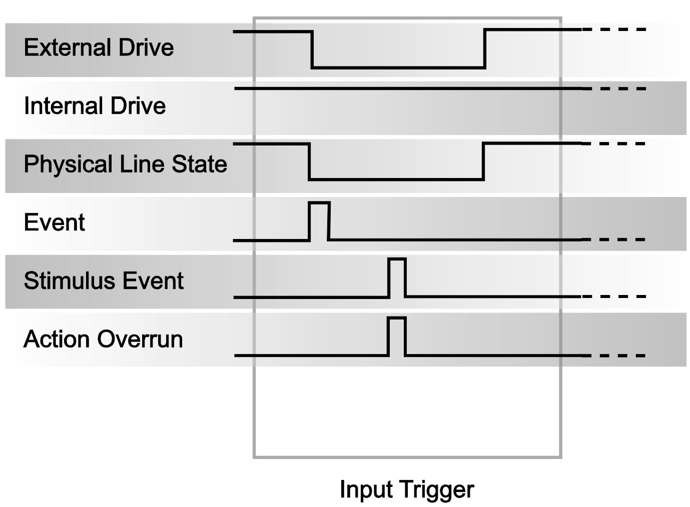
Falling edge trigger mode

The falling edge trigger mode generates low pulses and detects all falling edges. The figure titled "Falling edge input trigger" shows the characteristics of the falling edge input trigger; the figure titled "Falling edge output trigger" shows the falling edge output trigger.

Input characteristics:

- Detects all falling edges as input triggers.

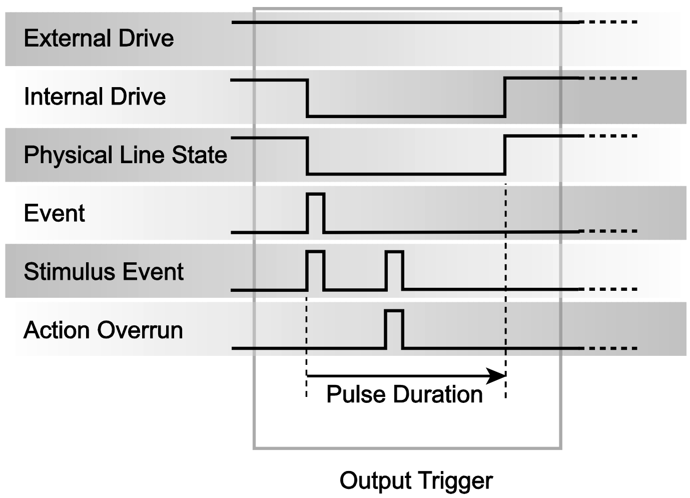
Figure 44: Falling edge input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse for the programmed pulse duration.
- An action overrun occurs if the physical line state is low and a source event occurs.
- When the trigger is asserted, it generates a low pulse for the programmed pulse duration.

Figure 45: Falling edge output trigger



Rising edge master trigger mode

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

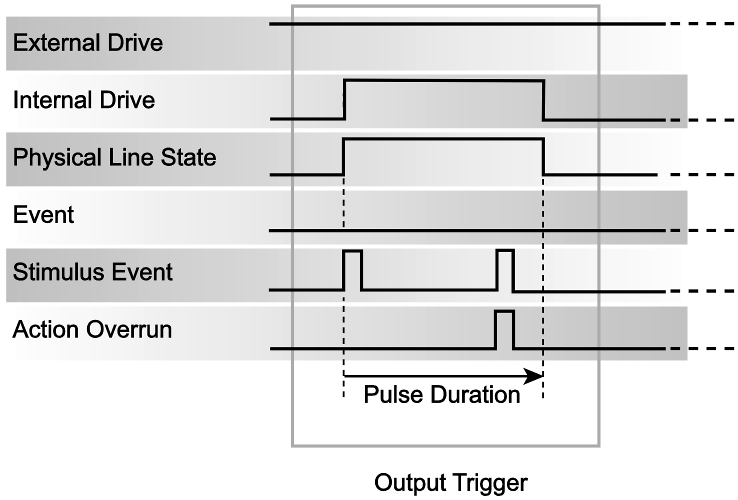
NOTE

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

Output characteristics:

- Configured trigger events, as well as the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands, cause the physical line state to float high during the trigger pulse duration.
- An action overrun occurs if the physical line state is high while a stimulus event occurs.
- When the trigger is asserted, it causes the physical line state to float high during the trigger pulse duration.

Figure 46: RisingM output trigger



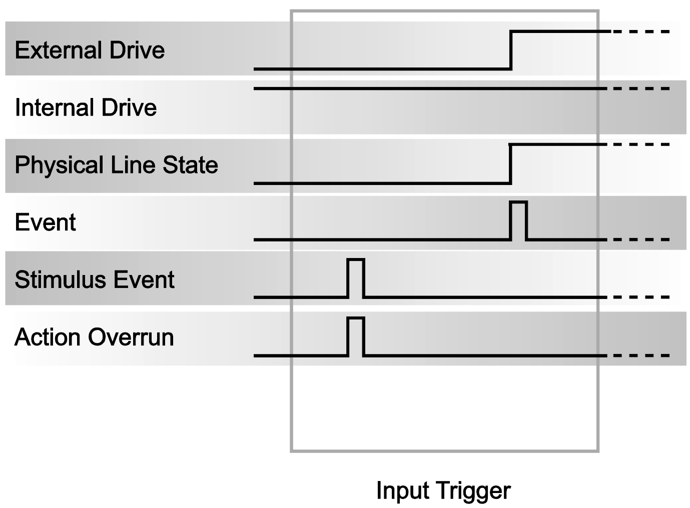
Rising edge acceptor trigger mode

The rising edge acceptor trigger mode (RisingA) generates a low pulse and detects rising edge pulses (see the following figures).

Input characteristics:

- All rising edges generate an input event.

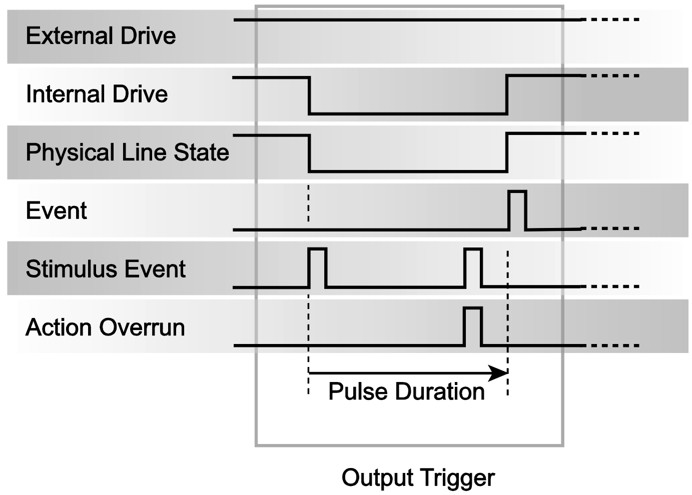
Figure 47: RisingA input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.

Figure 48: RisingA output trigger



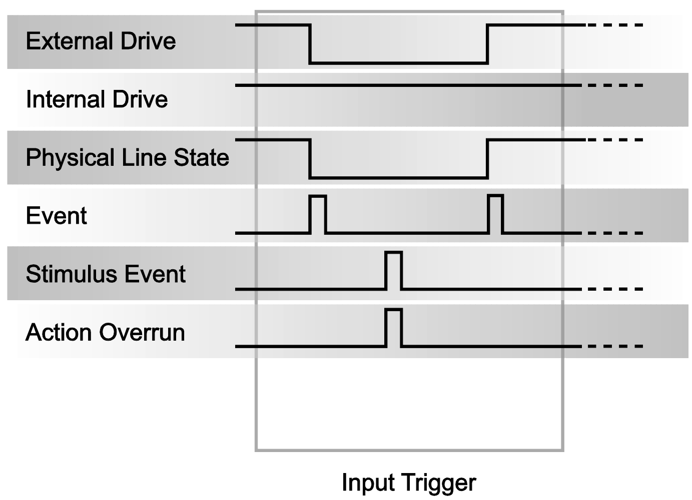
Either edge trigger mode

The either edge trigger mode generates a low pulse and detects both rising and falling edges.

Input characteristics:

- All rising or falling edges generate an input trigger event.

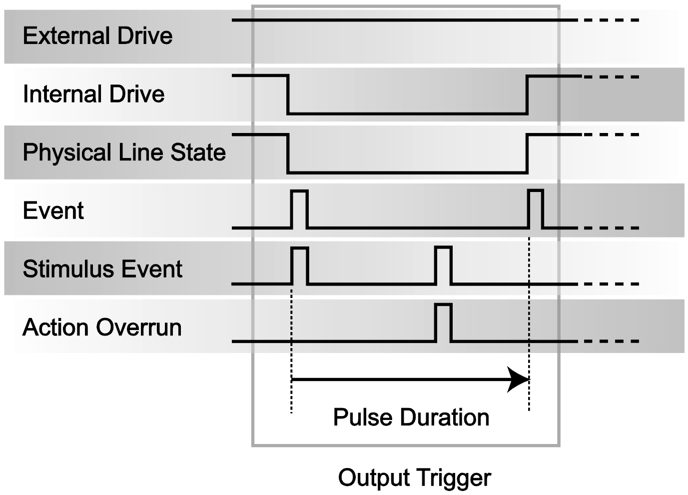
Figure 49: Either edge input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` commands generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low while a stimulus event occurs.

Figure 50: Either edge output trigger



Understanding synchronous triggering modes

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

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

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

Synchronous master trigger mode (SynchronousM)

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

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

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

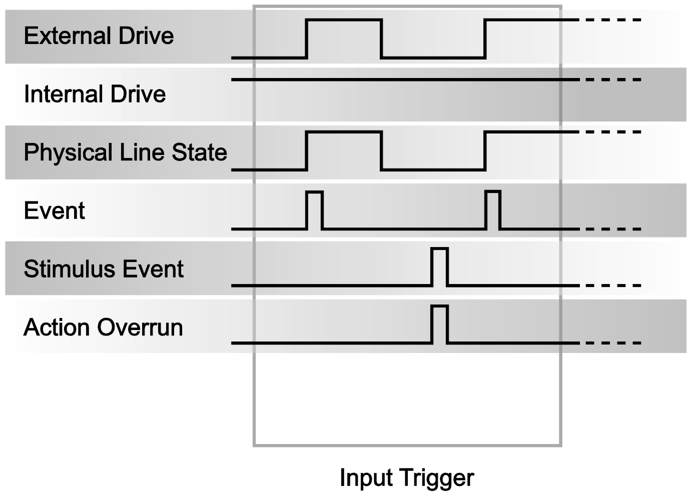
NOTE

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

Input characteristics:

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

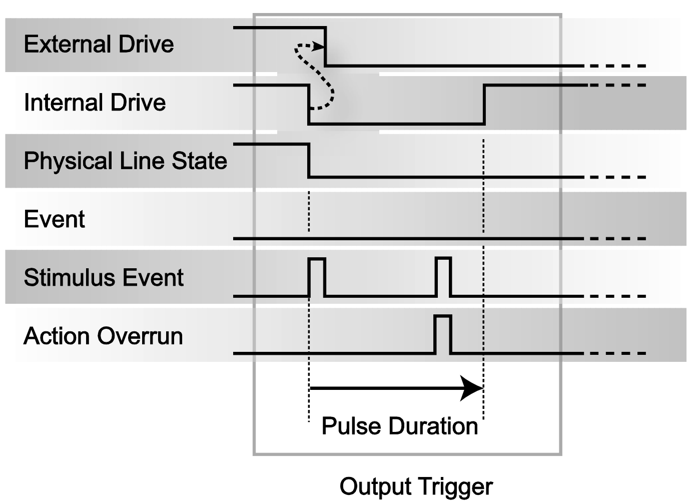
Figure 51: SynchronousM input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions generate a low pulse that is similar to the falling edge trigger mode.
- An action overrun occurs if the physical line state is low while a stimulus event occurs.
- When the trigger is asserted, it generates a low pulse that is similar to the Falling Edge trigger mode

Figure 52: SynchronousM output trigger



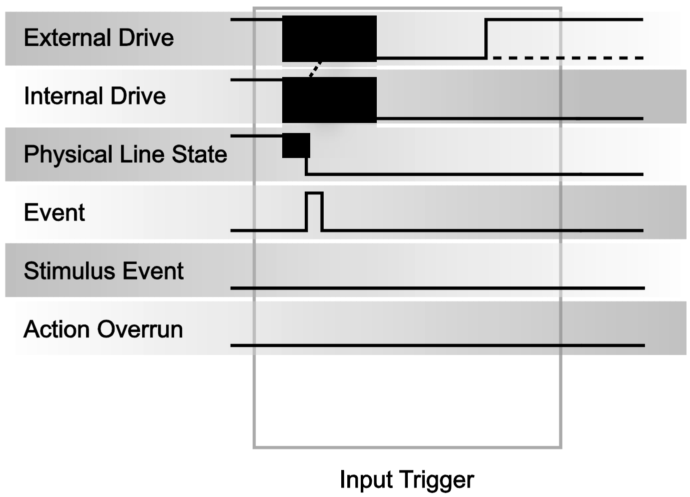
Synchronous acceptor trigger mode (SynchronousA)

Use the synchronous acceptor trigger mode (SynchronousA) in conjunction with the SynchronousM trigger mode. The roles of the internal and external drives are reversed in the SynchronousA trigger mode.

Input characteristics:

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

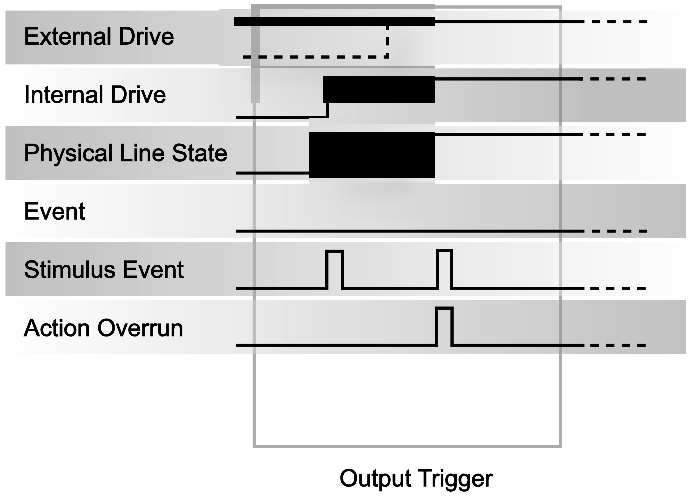
Figure 53: SynchronousA input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions release the line if the line is latched low. The pulse width is not used.
- When the trigger is asserted, it releases the line if the line is latched low.
- The physical line state does not change until all drives (internal and external) release the line.
- Action overruns occur if the internal drive is not latched low and a source event is received.

Figure 54: SynchronousA output trigger



Synchronous trigger mode

The synchronous trigger mode is a combination of SynchronousA and SynchronousM trigger modes.

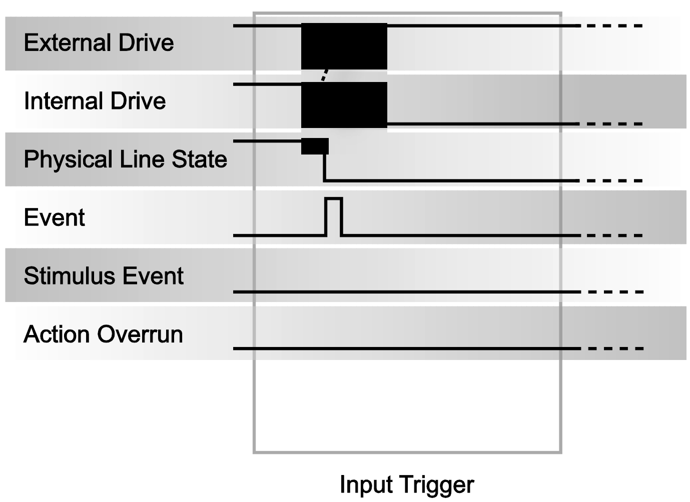
NOTE

Keithley Instruments recommends using SynchronousA and SynchronousM modes only.

Input characteristics:

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

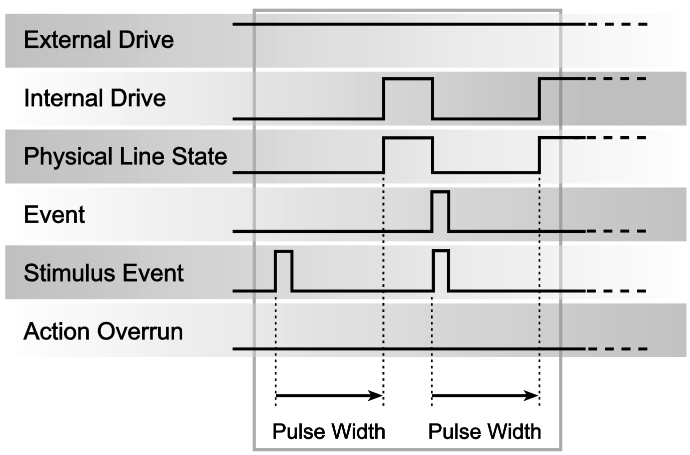
Figure 55: Synchronous input trigger



Output characteristics:

- In addition to trigger events from other trigger objects, the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions generate a low pulse for the programmed pulse duration if the line is latched low, a falling edge does not occur.
- When the trigger is asserted and the line is latched low, the pulse duration is enforced, and then the internal line drive is released.
- A normal falling edge pulse generates when the internal drive is not latched low and the `digio.trigger[N].assert()` and `tsplink.trigger[N].assert()` functions are issued.
- To mirror the SynchronousA trigger mode, set the pulse width to 1 μs or any small nonzero value.
- Action overruns are disabled.

Figure 56: Synchronous output trigger

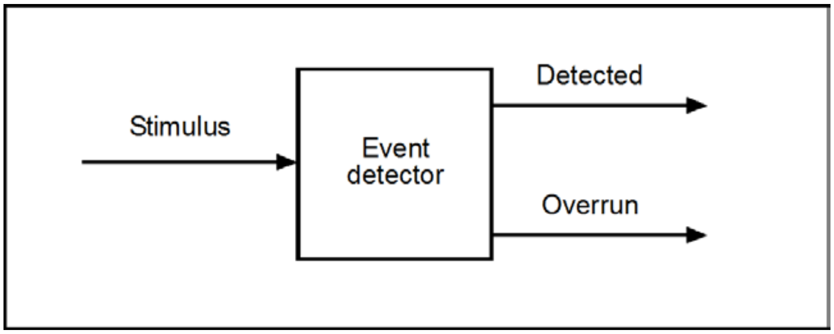


Output Triggers

Events

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

Figure 57: Event detector



Event blenders

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

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

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

trigger.blender[N].clear() (on page 8-422)
trigger.blender[N].orenable (on page 8-423)
trigger.blender[N].overrun (on page 8-424)
trigger.blender[N].stimulus[M] (on page 8-425)
trigger.blender[N].wait() (on page 8-427)

LXI Class B Triggering (IEEE-1588)

Introduction to IEEE-1588 based triggering

The Series 3700A uses IEEE-1588 precision time protocol (PTP) to implement synchronized measurements and initiate time-triggered events over the LAN (Ethernet) interface. IEEE-1588 is a requirement of the LXI B Functional Class. Using IEEE-1588, you can schedule instrument-driven actions, such as measurements, to occur at a specific date and time and synchronize timebases between instruments on the same network. You can only access these capabilities through the remote command interfaces.

NOTE

You can find detailed information on the syntax and usage of each remote command presented in this section in Command reference.

IEEE-1588 implementation in the Series 3700A

When you enable IEEE-1588 on a Series 3700A on a local network, the Series 3700A communicates with other IEEE-1588 enabled devices on the network through a dedicated network port called the PTP port. A predetermined algorithm then automatically selects the network device with the most accurate clock. This network device becomes the IEEE-1588 master. If multiple devices have the same clock accuracy, the protocol arbitrarily chooses one device to be the IEEE-1588 master.

When the protocol selects the Series 3700A as the master clock, the Series 3700A uses the time value stored in its battery-backed real-time clock and updates the time in all subordinate devices. When the protocol selects another networked device as the master clock, the Series 3700A is subordinate to the more accurate device and adjusts its time to that of the master clock. Additionally, the Series 3700A updates its battery-backed clock so that the time is "remembered" if the master clock is removed from the network.

At periodic intervals, the master clock synchronizes to all subordinate clocks through timestamped messages over the PTP port. This allows IEEE-1588 to maintain time synchronization between multiple devices on a network.

Program the synchronization interval in the Series 3700A using the ptp.syncinterval attribute The default synchronization interval is two seconds. Increasing the synchronization interval to values of more than two seconds increases the amount of time that it takes devices on the LAN to synchronize. If you change the synchronization interval, you must restart the clock of the Series 3700A by cycling its power.

Read the current time delay and offset between any subordinate device and its master on the LAN using the `ptp.ds.current` attribute. Synchronization of timestamps between IEEE-1588 enabled devices to within 150 ns can take as long as two minutes.

Correlating PTP to Coordinated Universal Time (UTC)

To ensure synchronization across networked devices, you must be aware of the time protocol utilized by those other devices on the network.

The most widely accepted time scale is Coordinated Universal Time (UTC); in many places, it is considered standard time. UTC is nearly the same time as Greenwich Mean Time (GMT), another very familiar time scale, and for the purposes of the Series 3700A, UTC and GMT are the same. Local time is offset from UTC according to time zones; additional offsets can occur due to Daylight Savings Time adjustments.

UTC suffers from discontinuities because of nonperiodic adjustments known as “leap seconds." These adjustments present problems because they can make events that occurred at different periods of time appear to occur at the same time. PTP is a time standard that does not have any discontinuities and has no adjustments for local time (that is, it is not time-zone aware). PTP is presented as the number of seconds since January 1, 1970.

The Series 3700A offers two versions of time for most IEEE-1588-related commands, `.seconds` and `.ptpseconds`, representing UTC and PTP respectively. IEEE-1588 requires that devices are synchronized using UTC or PTP time, not local time. The Series 3700A does not distinguish UTC, PTP, and local time; it is not time-zone aware. You must be aware of this when synchronizing with devices that are time-zone aware.

When IEEE-1588 selects a time-zone aware device to be the master clock, the Series 3700A accepts the time of that clock. This time may not agree with the local time of the Series 3700A, especially when a network spans multiple time zones. If you schedule events on the Series 3700A to occur according to your local time, events will not occur at the time you expect.

You can avoid confusion by setting the time on the Series 3700A to UTC time instead of local time. Manage the conversion from UTC to local time in your software application. For example, assume local time is Eastern Standard Time in the United States (EST), which is equivalent to GMT-5 (hours). Therefore, if the current local time is 3:00 PM, the UTC time is 8:00 PM. Set the time of the Series 3700A clock to 8:00 PM. If it is then synchronized with a time-zone aware master clock, its time will not change significantly.

NOTE

The Series 3700A does not differentiate UTC and PTP time. The `ptp.utcoffset` (on page 8-317) attribute is zero unless a master clock that is aware of the difference between UTC and PTP time populates this value. This value is volatile and does not persist through a power cycle.

Configuring and enabling IEEE-1588

To configure IEEE-1588, connect the Series 3700A to the LAN, along with any other IEEE-1588 enabled devices that you want to synchronize to the Series 3700A. Refer to the Series 3700A User Manual for information on connecting the Series 3700A to the LAN. If you want to synchronize multiple Series 3700A instruments on a LAN, each instrument must have the same PTP subdomain name.

The default PTP subdomain name is `_DFLT` for all Series 3700A devices. Use the ptp.subdomain attribute to change the subdomain name for any Series 3700A on the LAN. After changing the subdomain name, you must power cycle the Series 3700A to restart its clocks. If you have changed the subdomain name of any third-party IEEE-1588 enabled device in that subdomain, you must also restart its clock.

NOTE

Cycling the power to the Series 3700A does not return the IEEE-1588-related parameters to factory default state. To return these to factory defaults, perform a LAN configuration reset. This can be done using the lan.status.reset() function on the remote command interface. You can also perform a reset through the front-panel interface by entering the Main menu, selecting LAN, and selecting Reset.

Use the ptp.enable (on page 8-314) attribute to enable IEEE-1588 on the Series 3700A. The IEEE-1588 protocol then determines the master clock. The IEEE-1588 indicator on the front panel of the Series 3700A updates to display the IEEE-1588 status.

- If the indicator is off, IEEE-1588 is disabled or the device is not connected to a working network.
- If the network is not working, the LAN indicator blinks. If the indicator is solidly on, the IEEE-1588 is successfully enabled and synchronized, and the Series 3700A is a subordinate (slave) clock.
- If the indicator blinks once every second, IEEE-1588 is successfully enabled and synchronized, and the Series 3700A is the master clock.
- If the indicator blinks once every two seconds, IEEE-1588 is successfully enabled and synchronized, and the Series 3700A is the grandmaster clock.

You can also use the ptp.synchronized attribute to determine if the Series 3700A is a master or subordinate on the LAN.

NOTE

The ptp.enable (on page 8-314) attribute is saved in nonvolatile memory. Therefore, if you turn off a Series 3700A with IEEE-1588 enabled and then turn on the Series 3700A power on a different network, it attempts to synchronize with any other IEEE-1588 enabled devices on that new network. You do not need to re-enable IEEE-1588.

Monitoring alarms with LAN triggers and LXI event log

Use the LXI event log to monitor the firing of scheduled alarms. The LXI event log in the Series 3700A only captures LAN triggers that occur in its defined LXI domain. To monitor alarms, configure the alarm to generate a LAN trigger by using schedule.alarm[N].EVENT_ID (on page 8-351) as the control source for lan.trigger[N].stimulus (on page 8-288) in the trigger model. You can define up to eight LAN triggers.

Use lan.lxidomain (on page 8-273) to specify the LXI domain. Additionally, you can broadcast LAN triggers to all devices on an LXI domain, or you can transmit LAN triggers between two individual devices. To configure the LAN trigger broadcast, use lan.trigger[N].protocol (on page 8-287).

The following example demonstrates how to generate a LAN trigger when a scheduled alarm fires:

```
-- configure the LXI domain
lan.lxidomain=0
-- configure the LXI trigger to broadcast to all devices in this LXI domain
lan.trigger[2].protocol=2
lan.trigger[2].connect()
-- associate the firing of the alarm to the generation of a LAN trigger
lan.trigger[2].stimulus = schedule.alarm[1].EVENT_ID
```

LXI event log

The LXI event log of a Series 3700A monitors all LAN triggers that the instrument receives or generates. The LXI event log has nine comma-delimited fields. Below is an example entry to an LXI event log and a description of the log fields in order of appearance.

```
"17:26:35.690 10 Oct 2007, LAN0, 192.168.1.102, LXI, 0, 1192037132,
   1192037155.733269000, 0, 0x0"
```

Value	Description
`“17:26:35.690 10 Oct 2007”`	Formatted UTC time in 24-hour format including fractional seconds.
`“LAN0”`	Event identifier. This event identifier is zero-based (`LAN0`-`LAN7`). When specifying the LAN trigger using `lan.trigger[`*N*`]`, the minimum value for *N* is 1. Therefore `LAN0` to `LAN 7` corresponds to `lan.trigger[1]` through `lan.trigger[8]`, respectively.
`“192.168.1.102”`	IP address of the device that issued the LAN trigger.
`"LXI"`	LXI version identifier. Currently only LXI is defined.
`“0”`	LXI Domain number.
`“1192037132”`	Sequence number from the device that issued the LAN trigger. This number is incremented after generation of each LAN trigger.
`"1192037155.733269000”`	PTP time formatted as a floating point number.
`“0”`	The overflow from PTP seconds. Currently, this is “0”. Also referred to as IEEE-1588 Epoch.
`"0x0"`	Hex value of the flag field, which is the logical OR of several conditions (error=1, retransmission=2, hardware=4, acknowledgement=8).

Files

File formats

Each script, reading buffer, and saved setup is represented on a flash drive as a separate file.

Directories on a flash drive used with the Series 3700A can only contain a limited number of files. The top-level directory is limited to approximately 150 files, while subdirectories are limited to approximately 500 files. Once the limit has been reached, a "file system full" error message is generated.

Default file extensions

You must specify the full filename, including the extension, when sending commands. Note, however, that the front panel automatically generates a generic filename that you can use as a base for naming your files. Also, some commands (for example, io.open() (on page 8-264)) will work with either a relative or absolute path to the current working directory.

The Model 3706A has the following set of default extensions:

- **.tsp** (Test Script Processor) for scripts
- **.csv** (comma-separated values) for reading buffers
- **.set** for saved setups

File system navigation

The Lua FS library provides the command set necessary to navigate the file system and list the available files on a flash drive. The instrument encapsulates this command set as an `fs` logical instrument, so that the file system of any given node is available to the entire TSP-Link® system. For example, the command `node[5].fs.readdir(".")` can be used to read the contents of the current working directory on Node 5.

To allow for future enhancements, the root folder of the USB flash drive has the absolute path `/usb1/`.

NOTE

Both slash (/) and backslash (\) are supported as directory separators, but because backslash is an escape character in Lua, it appears as a double backslash in this context.

The following Lua FS commands, which support basic navigation and directory listing, are included for your reference.

- fs.chdir() (on page 8-258)
- fs.cwd() (on page 8-258)
- fs.is_dir() (on page 8-258)
- fs.is_file() (on page 8-259)
- fs.mkdir() (on page 8-259)
- fs.readdir() (on page 8-260)
- fs.rmdir() (on page 8-260)

The following Lua FS commands are not supported at this time:

- fs.chmod
- fs.chown
- fs.stat

File I/O

You can use the file I/O commands to open and close directories and files, write data, or to read a file on an installed USB flash drive. File I/O commands are organized into two groups:

- Commands that reside in the `fs` and `io` table, for example: `io.open()`, `io.close()`, `io.input()`, and `io.output()`. Use these commands to manage file system directories; open and close file descriptors; and perform basic I/O operations on a pair of default files (one input and one output).
- Commands that reside in the file descriptors (for example: *`fileVar`*`:seek()`, *`fileVar`*`:write()`, and *`fileVar`*`:read()`) operate exclusively on the file with which they are associated.

The root folder of the USB flash drive has the absolute path:

`"/usb1/"`

NOTE

You can use either the slash (`/`) or backslash (`\`) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you will generally need to use a double backslash (`\\`) when you are creating scripts or sending commands to the instrument.

For basic information about navigation and directory listing of files on a flash drive, see File system navigation (on page 6-12).

File descriptor commands for file I/O use a colon (:) to separate the command parts rather than a period (.), like the `io` commands.

File descriptors cannot be passed between nodes in a TSP-Link® system, so the `io.open()`, `fileVar::read()`, and `fileVar::write` commands are not accessible to the TSP-Link system. However, the default input and output files mentioned above allow for the execution of many file I/O operations without any reference to a file descriptor.

fileVar:close() (on page 8-252)
fileVar:flush() (on page 8-252)
fileVar:read() (on page 8-253)
fileVar:seek() (on page 8-254)
fileVar:write() (on page 8-254)
fs.chdir() (on page 8-258)
fs.cwd() (on page 8-258)
fs.is_dir() (on page 8-258)
fs.is_file() (on page 8-259)
fs.mkdir() (on page 8-259)
fs.readdir() (on page 8-260)
fs.rmdir() (on page 8-260)
io.close() (on page 8-262)
io.flush() (on page 8-262)
io.input() (on page 8-263)
io.open() (on page 8-264)
io.output() (on page 8-264)
io.read() (on page 8-265)
io.type() (on page 8-266)
io.write() (on page 8-266)
os.remove()
os.rename()

The following standard I/O commands are not supported at this time:

File	I/O
• *fileVar*:lines() • *fileVar*:setvbuf()	• io.lines() • io.popen()

Script examples

The following script will open three different files to help illustrate the differences between the `io` commands and file descriptor commands. After opening the files, the script designates each one as the default output file (using the `io.output` command). While each file is the default for file writes (using the `io.write` command), the script also uses the file descriptor from the `io.open` to write to the file (`file:write` command).

After all files are closed (using the `io.close` command), the script will open the files again for reading. Two files are read by:

- Designating the file the default input file (using the `io.input` command)
- Being the default read contents of file (using the `io.read` command)

The third file is read by using the file descriptor from the open (`file:read` command). After reading all files, they are closed using the file descriptor and close option (`file:close` command).

```
loadscript file_io_test
   -- get the current date and time
   date_time = os.date("%c", os.time())
   -- open the three files for writing
   myfile1, myfile1_err, myfile1_errnum = io.open("/usb1/myfile_io1", "w")
   myfile2, myfile2_err, myfile2_errnum  = io.open("/usb1/myfile_io2", "w")
   myfile3, myfile3_err, myfile3_errnum  = io.open("/usb1/myfile_io3", "w")
   if (io.type(myfile1) == "file") then
         if (io.type(myfile2) == "file") then
            if (io.type(myfile3) == "file") then
               -- make myfile1 the default output file
               io.output(myfile1)
               -- write some data to the default file
               io.write("Using io write to myfile1 to io output\n")
               io.write(date_time)
               io.write("\n")
               -- now write to myfile2 using descriptor rather than io write
   command
               myfile2:write("    file handle to write to myfile2\n")
               myfile2:write("    while myfile1 is output file for io\n")
               -- make myfile2 the default output file
               io.output(myfile2)
               -- write some data to the default file
               io.write("Using io write to myfile2 to io output\n")
               io.write(date_time)
               io.write("\n")
               -- now write to myfile3 using descriptor rather than io write
   command
               myfile3:write("    file handle to write to myfile3\n")
               myfile3:write("    while myfile2 is output file for io\n")
               -- make myfile3 the default output file
               io.output(myfile3)
               -- write some data to the default file
               io.write("Using io write to myfile3 to io output\n")
               io.write(date_time)
               io.write("\n")
               -- now write to myfile1 using descriptor rather than io write
   command
               myfile1:write("    file handle to write to myfile1\n")
               myfile1:write("    while myfile3 is output file for io\n")
               -- use the io close rather than file descriptor close command
               io.close(myfile1)
               io.close(myfile2)
               io.close(myfile3)
         else
               print("myfile3 did not open for write")
               print("error string is " .. myfile3_err)
               print("error number is " .. myfile3_errnum)
         end
      else
         print("myfile2 did not open for write")
         print("error string is " .. myfile2_err)
         print("error number is " .. myfile2_errnum)
      end
   else
      print("myfile1 did not open for write")
      print("error string is " .. myfile1_err)
```

```
      print("error number is " .. myfile1_errnum)
   end
   -- open the 3 files again for reading
   myfile1, myfile1_err, myfile1_errnum = io.open("/usb1/myfile_io1", "r")
   myfile2, myfile2_err, myfile2_errnum  = io.open("/usb1/myfile_io2", "r")
   myfile3, myfile3_err, myfile3_errnum  = io.open("/usb1/myfile_io3", "r")
   if (io.type(myfile1) == "file") then
         if (io.type(myfile2) == "file") then
         if (io.type(myfile3) == "file") then
            -- make myfile1 the default input file
            io.input(myfile1)
            -- read the default file
            filecontents = io.read("*a")
            print("contents of myfile1 are:")
            print(filecontents)
            print()
            -- make myfile2 the default input file
            io.input(myfile2)
            -- read the default file
            filecontents = io.read("*a")
            print("contents of myfile2 are:")
            print(filecontents)
            print()
            -- read myfile3 using file descriptor instead of io read
            filecontents = myfile3:read("*a")
            print("contents of myfile3 are:")
            print(filecontents)
            print()
            -- use file descriptor close command rather than io close
               myfile1:close()
               myfile2:close()
               myfile3:close()
            else
               print("myfile3 did not open for read")
               print("error string is " .. myfile3_err)
               print("error number is " .. myfile3_errnum)
            end
         else
            print("myfile2 did not open for read")
            print("error string is " .. myfile2_err)
            print("error number is " .. myfile2_errnum)
         end
   else
         print("myfile1 did not open for read")
      print("error string is " .. myfile1_err)
      print("error number is " .. myfile1_errnum)
   end
endscript
```

After downloading the above script, type `file_io_test()` to execute the script:

```
file_io_test()
```

The following output is returned after executing the `file_io_test()` script:

```
contents of myfile1 are:
Using io write to myfile1 to io output
11/27/07 07:57:23
   file handle to write to myfile1
   while myfile3 is output file for io

contents of myfile2 are:
   file handle to write to myfile2
   while myfile1 is output file for io
Using io write to myfile2 to io output
11/27/07 07:57:23

contents of myfile3 are:
   file handle to write to myfile3
   while myfile2 is output file for io
Using io write to myfile3 to io output
11/27/07 07:57:23
```

The following script will open a file called `myfiletest` three times. The first time it is opened is for writing. Note that opening an existing file for writing deletes any existing information in the file. The second time it is opened is for appending more data to the existing data in the file. Opening a file for append will not delete any existing data; it only adds data to the end of the existing file contents. The third time the file is opened is for reading the entire contents of the file (existing data and appended data).

```
loadscript filetest
      -- script to write 2 lines to a file
      -- append 2 lines to the same file
      -- read the entire file contents and print them

      -- open the file for writing
      myfile = io.open("/usb1/myfiletest", "w")
      if io.type(myfile) == "file" then
         myfile:write("This is my first line WRITING\n")
         myfile:write("This is my next line WRITING\n")
         myfile:close()

         -- open the file for appending
         myfile = io.open("/usb1/myfiletest", "a")
         if io.type(myfile) == "file" then
            myfile:write("This is my first APPEND line\n")
            myfile:write("This is my next APPEND line\n")
            myfile:close()
            -- open the file for reading
            myfile = io.open("/usb1/myfiletest", "r")
            if io.type(myfile) == "file" then
               filecontents = myfile:read("*a")
               print("the file contains:")
               print()
               print(filecontents)
               myfile:close()
            else
               print("The file did not open correctly for reading")
             end
         else
            print("The file did not open correctly for appending")
      end
   else
         print("The file did not open correctly for writing")
   end
endscript
```

After downloading the above script, type `filetest()` to execute the script. Here are the output results:

```
the file contains:
This is my first line WRITING
This is my next line WRITING
This is my first APPEND line
This is my next APPEND line
```

Display operations

Display functions and attributes

You will use the display functions and attributes to perform the display operations covered in this section. The following table lists each display function or attribute (in alphabetic order) and cross references it to the section topic where the function or attribute is explained.

TSP command reference (on page 8-1) provides additional information about the display functions and attributes.

Cross-referencing functions and attributes to section topics

Function or attribute	Section topic
`display.clear()`	Clearing the display (on page 3-33)
`display.getannunciators()`	Indicators (on page 3-38)
`display.getcursor()`	Cursor position (on page 3-33)
`display.getlastkey()`	Capturing key-press codes (on page 3-42)
`display.gettext()`	Displaying text messages (on page 3-34)
`display.inputvalue()`	Parameter value prompting (on page 3-37)
`display.loadmenu.add()` `display.loadmenu.delete()`	Load test menu (on page 3-40)
`display.locallockout`	LOCAL lockout (on page 3-39)
`display.menu()`	Menu (on page 3-36)
`display.prompt()`	Parameter value prompting (on page 3-37)
`display.screen`	Display screen
`display.sendkey()`	Sending key codes (on page 3-42)
`display.setcursor()`	Cursor position (on page 3-33)
`display.settext()`	Displaying text messages (on page 3-34)

Display messages

NOTE

Most of the display functions and attributes that are associated with display messaging will automatically select the user screen. The attribute for the display screen is explained in Display screen.

The `reset()` function has no effect on the defined display message or its configuration, but will set the display mode back to the previous display mode.

The display of the Series 3700A can be used to display user-defined messages. For example, while a test is running, the following message can be displayed on the Series 3700A.

```
Test in Process
Do Not Disturb
```

The top line of the display can accommodate up to 20 characters (including spaces). The bottom line can display up to 32 characters (including spaces) at a time.

NOTE

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions (which are explained in the following paragraphs) are overlapped, nonblocking commands. The script will NOT wait for one of these commands to complete.

These nonblocking functions do not immediately update the display. For performance considerations, they write to a shadow and will update the display as soon as processing time becomes available.

Clearing the display

When sending a command to display a message, a previously defined user message is not cleared. The new message starts at the end of the old message on that line. It is good practice to routinely clear the display before defining a new message.

After displaying an input prompt, the message will remain displayed even after the operator performs the prescribed action. The `clear()` function must be sent to clear the display. To clear both lines of the display, but not affect any of the indicators, send the following function:

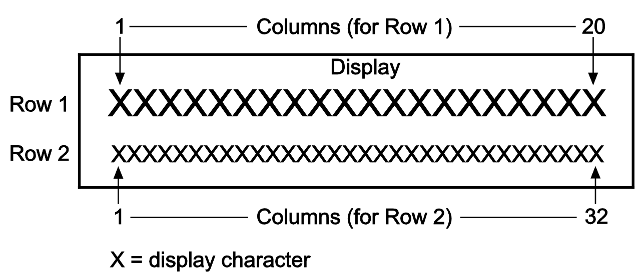
```
display.clear()
```

Cursor position

When displaying a message, the cursor position determines where the message will start. On power-up, the cursor is positioned at row 1, column 1 (see the following figure). At this cursor position, a user-defined message will be displayed on the top row (row 1).

Top line text will not wrap to the bottom line of the display automatically. Any text that does not fit on the current line will be truncated. If the text is truncated, the cursor will be left at the end of the line.

Figure 58: Row/column format for display messaging



The function to set cursor position can be used two ways:

```
display.setcursor(row, column)
display.setcursor(row, column, style)
```

Where:

row	`1` or `2`
column	`1` to `20` (row 1) `1` to `32` (row 2)
style	`0` (invisible) `1` (blink)

When set to 0, the cursor will not be seen. When set to 1, a display character will blink to indicate its position.

The `display.getcursor()` function returns the present cursor position, and can be used three ways:

```
row, column, style = display.getcursor()
row, column = display.getcursor()
row = display.getcursor()
```

The following programming example illustrates how to position the cursor on row 2, column 1, and then read the cursor position:

```
display.setcursor(2, 1)
row, column = display.getcursor()
print(row, column)
```

Output:

```
2.00000e+00 1.00000e+00
```

Displaying text messages

To define and display a message, use the `display.settext(text)`function (*text* is the text string to be displayed). The message will start at the present cursor position. The following programming example illustrates how to display "Test in Process" on the top line, and "Do Not Disturb" on the bottom line:

```
display.clear()
display.setcursor(1, 1, 0)
display.settext("Test in Process")
display.setcursor(2, 6, 0)
display.settext("Do Not Disturb")
```

Character codes

The following special codes can be embedded in the `text` string to configure and customize the message:

- `$N` Starts text on the next line (newline). If the cursor is already on line 2, text will be ignored after the '`$N`' is received.
- `$R` Sets text to Normal.
- `$B` Sets text to Blink.
- `$D` Sets text to Dim intensity.
- `$F` Set text to background blink.
- `$$` Escape sequence to display a single "$".

In addition to displaying alphanumeric characters, you can display other special characters. Refer to Display character codes for a complete listing of special characters and their corresponding codes. The following programming example illustrates how to display the Greek symbol omega (Ω) :

```
display.clear()
c = string.char(18)
display.settext(c)
```

The following programming example illustrates how to use the `$N` and `$B` character codes to display the message "Test in Process" on the top line and the blinking message "Do Not Disturb" on the bottom line:

```
display.clear()
display.settext("Test in Process $N$BDo Not Disturb")
```

The following programming example illustrates how to use the `$$` character code to display the message "You owe me $8" on the top line:

```
display.clear()
display.setcursor(1, 1)
display.settext("You owe me $$8")
```

If the extra `$` character is not included, the `$8` would be interpreted as an undefined character code and will be ignored. The message "You owe me" will instead be displayed.

NOTE

Be careful when embedding character codes in the text string; it is easy to forget that the character following the `$` is part of the code. For example, assume you want to display "Hello" on the top line and "Nate" on the bottom line, and so you send the following command:

```
display.settext("Hello$Nate")
```

The above command displays "Hello" on the top line and "ate" on the bottom line. The correct syntax for the command is as follows:

```
display.settext("Hello$NNate")
```

Returning a text message

The `display.gettext()` function returns the displayed message (*text*) and can be used in five ways:

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

Where:

embellished	Returns text as a simple character string (`false`) or includes character codes (`true`)
row	The row to read text from (`1` or `2`); if not included, text from both rows is read
columnStart	Starting column for reading text
columnEnd	Ending column for reading text

Sending the command without the *row* parameter returns both lines of the display. The $N character code will be included to show where the top line ends and the bottom line begins. The $N character code will be returned even if *embellished* is set to false.

With *embellished* set to true, all other character codes that were used in the creation of each message line will be returned along with the message. With *embellished* set to false, only the message will be returned.

Sending the command without the *columnStart* parameter defaults to column 1. Sending the command without the *columnEnd* argument defaults to the last column (column 20 for row 1, column 32 for row 2).

Input prompting

Display messaging can be used along with front panel controls to make a user script interactive. In an interactive script, input prompts are displayed so that the operator can perform a prescribed action using the front panel controls. While displaying an input prompt, the test will pause and wait for the operator to perform the prescribed action from the front panel.

Menu

A user-defined menu can be presented on the display. The menu consists of the menu name on the top line, and a selectable list of menu items on the bottom line. To define a menu, use the `display.menu(`*menu*`, `*items*`)` function.

Where:

menu	The name of the menu; use a string of up to 20 characters (including spaces)
items	A string is made up of one or more menu items; each item must be separated by white space

When the `display.menu()` function is sent, script execution waits for the operator to select one of the menu items. Rotate the navigation wheel to place the blinking cursor on a menu item. Items that do not fit in the display area are displayed by rotating the navigation wheel to the right. With the cursor on the menu item, press the navigation wheel (or the **ENTER** key) to select it.

Pressing the EXIT (LOCAL) key does not abort the script while the menu is displayed, but it will return `nil`. The script can be aborted by calling the `exit()` function when `nil` is returned.

The following programming example illustrates how to present the operator with the choice of two menu items: Test1 or Test2. If Test1 is selected, the message `Running Test1` is displayed. If Test2 is selected, the message `Running Test2` is displayed.

```
display.clear()
menu = display.menu("Sample Menu", "Test1 Test2")
if menu == "Test1" then
   display.settext("Running Test1")
else
   display.settext("Running Test2")
end
```

Parameter value prompting

There are two functions that you can use to create an editable input field on the user screen at the present cursor position: `display.inputvalue()` and `display.prompt()`.

The `display.inputvalue()` function uses the user screen at the present cursor position. Once the command is finished, it returns the user screen to its previous state. The `display.prompt()` function creates a new edit screen and does not use the user screen.

Each of these two functions can be used in four ways:

```
display.inputvalue(format)
display.inputvalue(format, default)
display.inputvalue(format, default, min)
display.inputvalue(format, default, min, max)
display.prompt(format, units, help)
display.prompt(format, units, help, default)
display.prompt(format, units, help, default, min)
display.prompt(format, units, help, default, min, max)
```

Where:

format	String that creates an editable input field on the user screen at the present cursor position (examples: `+0.00 00`, `+00`, `0.00000E+0`) **Value field:** **+** = Include for positive/negative value entry; omitting the + prevents negative value entry **0** = Defines the digit positions for the value (up to six zeros (0)) **Exponent field (optional):** **E** = include for exponent entry **+** = Include for positive/negative exponent entry; omitting the + prevents negative value entry **0** = Defines the digit positions for the exponent
default	Option to set a default value for the parameter, which will be displayed when the command is sent
min	Option to specify minimum limits for the input field • When NOT using the "+" sign for the value field, the minimum limit cannot be set to less than zero • When using the "+" sign, the minimum limit can be set to less than zero (for example, `-2`)
max	Option to specify maximum limits for the input field
units	Text string to identify the units for the value (8 characters maximum), for example: Units text is "V" for volts and "A" for amperes
help	Informational text string to display on the bottom line (32 characters maximum).

Both the `display.inputvalue()` and `display.prompt()` functions display the editable input field, but the `display.inputvalue()` function does not include the text strings for *units* and *help*.

After one of the above functions is executed, command execution will pause and wait for the operator in input the source level. The program will continue after the operator enters the value by pressing the navigation wheel ◷ or the ENTER key.

In this example, the command `display.prompt` prompts the operator to input a measurement speed. If the operator does not enter a value, the default level of 1 is set when the operator presses **ENTER**. The operator must input values that are within the limits (minimum of 0.01 and maximum of 3.0); any other values are not accepted.

Example: Interactive script

Code	Output
`myFunc = display.menu ("Select function", "dcvolts twowireohms")`	Prompt operator to select function.
`if (myFunc == "dcvolts") then myRange = display.menu("Select range", "10 100") if (myRange == "10") then rangeValue = 10 else rangeValue = 100 end else myRange = display.menu("Select range", "1000 10000") if (myRange == "1000") then rangeValue = 1000 else rangeValue = 10000 end end`	Prompt for range based on function selected.
`speed = display.prompt("0.00", "NPLC", "Enter measure speed", 1, 0.01, 3)`	Prompt operator to set the measurement speed.
`dmm.reset("all") dmm.func = myFunc dmm.range = rangeValue dmm.nplc = speed print(dmm.measure())`	Wait for operator to set the measurement speed.

```
myFunc = display.menu ("Select function",
  "dcvolts twowireohms")
if (myFunc == "dcvolts") then
   myRange = display.menu("Select range",
    "10 100")
   if (myRange == "10") then
      rangeValue = 10
   else
      rangeValue = 100
   end
else
   myRange = display.menu("Select range",
    "1000 10000")
   if (myRange == "1000") then
      rangeValue = 1000
   else
      rangeValue = 10000
   end
end
speed = display.prompt("0.00", "NPLC",
    "Enter measure speed", 1, 0.01, 3)
dmm.reset("all")
dmm.func = myFunc
dmm.range = rangeValue
dmm.nplc = speed
print(dmm.measure())
```

Display trigger wait and clear

The `display.trigger.wait()` function causes the instrument to wait for the front panel TRIG key to be pressed, while the `display.trigger.clear()` function clears the trigger event detector.

Indicators

To determine which display indicators are turned on, use the `display.getannunciators()` function. For example, to determine which display indicators are turned on, send the following commands.

```
annun = display.getannunciators()
print(annun)
```

The 16-bit binary equivalent of the returned value is a bitmap. Each bit corresponds to an indicator. If the bit is set to “1”, the indicator is turned on. If the bit is set to “0”, the indicator is turned off.

The following table identifies the bit position for each indicator. The table also includes the weighted value of each bit. The returned value is the sum of all the weighted values for the bits that are set.

For example, assume the returned bitmap value is 34061. The binary equivalent of this value is as follows:

1000010100001101

For the above binary number, the following bits are set to "1": 16, 11, 9, 4, 3 and 1. Using the table, the following indicators are on: REL, REM, EDIT, AUTO, 4W and FILT.

Bit identification for indicators

Bit	B16	B15	B14	B13	B12	B11	B10	B9
Annunciator	`REL`	`REAR`	`SRQ`	`LSTN`	`TALK`	`REM`	`ERR`	`EDIT`
Weighted value*	32768	16384	8192	4096	2048	1024	512	256
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

Bit	B8	B7	B6	B5	B4	B3	B2	B1
Annunciator	`SMPL`	`STAR`	`TRIG`	`ARM`	`AUTO`	`4W`	`MATH`	`FILT`
Weighted value*	128	64	32	16	8	4	2	1
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

* The weighted values are for bits that are set to "1." Bits set to "0" have no value.

Not all of the above indicators shown in above table may be used by the Series 3700A.

Local lockout

You can use the front-panel EXIT (LOCAL) key to cancel remote operation and return control to the front panel. However, this key can be locked-out to prevent a test from being interrupted. When locked, this key becomes a NO-OP (no operation). Configure the following attribute to lock or unlock the EXIT (LOCAL) key:

`display.locallockout =` *`lockout`*

Where *`lockout`* is set to one of the following values:

- `0` or `display.UNLOCK`
- `1` or `display.LOCK`

For example, to lock out the EXIT (LOCAL) key:

```
display.locallockout = display.LOCK
```

Load test menu

Allows you to run scripts and code from the front panel that you created through the communication interface, or configuration scripts created by pressing the front-panel MENU key, then selecting SCRIPT > CREATE-CONFIG.

To open this menu, press **LOAD**.

The User option loads code that was added to Load Test with the display.loadmenu.add() (on page 8-138) command.

The Scripts option loads named scripts that were added to the runtime environment. See Manage scripts (on page 7-3) for information on creating and loading scripts.

After selecting code or script from the User or Scripts option, you can press **RUN** to execute the selected code or script.

User tests

User tests can be added to or deleted from the USER TESTS submenu.

Adding USER TESTS menu entries

You can use the following function in either of two ways to add an entry into the USER TESTS menu:

```
display.loadmenu.add(displayname, code)
display.loadmenu.add(displayname, code, memory)
```

Where:

displayname	The name string that is added to the USER TESTS menu.
code	The code that is run from the USER TESTS menu when the RUN button is pressed. It can include any valid Lua code.
memory	A value that specifies if the *code* and *displayname* parameters are saved in nonvolatile memory. Set to one of the following values: `0` or `display.DONT_SAVE` `1` or `display.SAVE` (this is the default setting)

Scripts, functions, and variables that are used in the *code* are not saved when `display.SAVE` is used. Functions and variables need to be saved with the script (see Manage scripts (on page 7-3)). If the script is not saved in nonvolatile memory, it is lost when the Series 3700A is turned off. See Example 1 below.

Example 1:

Assume a script with a function named "DUT1" has been loaded into the Series 3700A, and the script has not been saved in nonvolatile memory.

Now assume you want to add a test named "Test" to the USER TESTS menu. You want the test to run the function named "DUT1" and sound the beeper. The following programming example illustrates how to add "Test" to the menu, define the *code*, and then save *displayname* and *code* in nonvolatile memory:

```
display.loadmenu.add("Test", "DUT1() beeper.beep(2, 500)", display.SAVE)
```

When "Test" is run from the front-panel USER TESTS menu, the function named "DUT1" executes and the beeper beeps for two seconds.

Now assume you turn the Series 3700A power off and then on again. Because the script was not saved in nonvolatile memory, the function named "DUT1" is lost. When "Test" is again run from the front panel, the beeper beeps, but "DUT1" will not execute because it no longer exists in the run-time environment.

Example 2:

The following command adds an entry called "Part1" to the front-panel USER TESTS submenu for the code "testpart([[Part1]], 5.0)", and saves it in nonvolatile memory:

```
display.loadmenu.add("Part1", "testpart([[Part1]], 5.0)", display.SAVE)
```

Deleting USER TESTS menu entries

You can use the following function to delete an entry from the front-panel USER TESTS menu:

`display.loadmenu.delete(`*`displayname`*`)`

Where:

displayname	Name to delete from the menu.

The following programming example removes the entry named "Part1" from the front-panel USER TESTS menu:

```
display.loadmenu.delete("Part1")
```

LOAD TEST menu options

Allows you to run scripts and code from the front panel that you created through the communication interface, or configuration scripts created by pressing the front-panel MENU key, then selecting SCRIPT > CREATE-CONFIG.

To open this menu, press **LOAD**.

The User option loads code that was added to Load Test with the display.loadmenu.add() (on page 8-138) command.

The Scripts option loads named scripts that were added to the run-time environment. See Manage scripts (on page 7-3) for information on creating and loading scripts.

After selecting code or script from the User or Scripts option, you can press **RUN** to execute the selected code or script.

Key-press codes

Sending key codes

Key codes are provided to remotely simulate pressing a front-panel key or the navigation wheel. There are also key codes to simulate rotating the navigation wheel to the left or right (one click at a time). Use the `display.sendkey()` function to perform these actions. The following programming examples illustrate how to simulate pressing the MENU key in two different ways:

```
display.sendkey(display.KEY_MENU)
display.sendkey(68)
```

Capturing key-press codes

A history of the key code for the last pressed front panel key is maintained by the Series 3700A. When the instrument is turned on (or when transitioning from local to remote operation), the key code is set to `0` (`display.KEY_NONE`).

When a front-panel key is pressed, the key code value for that key can be captured and returned. There are two functions associated with the capture of key-press codes: `display.getlastkey()` and `display.waitkey()`.

display.getlastkey()

The `display.getlastkey()` function is used to immediately return the key code for the last pressed key. The following programming example illustrates how to display the last key pressed:

```
key = display.getlastkey()
print(key)
```

The above code will return the key code value (see the following table). Remember that a value of `0` (`display.KEY_NONE`) indicates that the key code history had been cleared.

Key code values returned for display.getlastkey

Value	Key list	Value	key list
0	(display.KEY_NONE)	82	(display.KEY_ENTER)
65	(display.KEY_RANGEUP)	83	(display.KEY_MEASB)
67	(display.KEY_RELB)	84	(display.DIGITSB)
68	(display.KEY_MENU)	85	(display.KEY_RECALL)
69	(display.KEY_MODEA)	86	(display.KEY_MEASA)
70	(display.KEY_RELA)	87	(display.KEY_DIGITSA)
71	(display.KEY_RUN)	90	(display.KEY_LIMITB)
72	(display.KEY_DISPLAY)	91	(display.KEY_SPEEDB)
73	(display.KEY_AUTO)	92	(display.KEY_TRIG)
74	(display.KEY_FILTERB)	93	(display.KEY_LIMITA)
75	(display.KEY_EXIT)	94	(display.KEY_SPEEDA)
76	(display.KEY_SRCB)	95	(display.KEY_LOAD)
77	(display.KEY_FILTERA)	97	(display.WHEEL_ENTER)
78	(display.KEY_STORE)	103	(display.KEY_RIGHT)
79	(display.KEY_SRCA)	104	(display.KEY_LEFT)
80	(display.KEY_CONFIG)	114	(display.WHEEL_RIGHT)
81	(display.KEY_RANGEDOWN)		

display.waitkey()

The `display.waitkey()` function captures the key code value for the next key press:

```
key = display.waitkey()
```

After sending the `display.waitkey()` function, the script will pause and wait for the operator to press a front-panel key. For example, if the MENU key is pressed, the function will return the value 68, which is the key code for that key. The key code values are the same as listed in display.getlastkey() (on page 8-134).

The following programming example illustrates how to prompt the user to press the EXIT (LOCAL) key to abort the script, or any other key to continue it:

```
display.clear()
display.setcursor(1, 1)
display.settext("Press EXIT to Abort")
display.setcursor(2, 1)
display.settext("or any key to continue")
key = display.waitkey()
display.clear()
display.setcursor(1, 1)
if key == 75 then
   display.settext("Test Aborted")
   exit()
else
   display.settext("Test Continuing")
end
```

The above code captures the key that is pressed by the operator. The key code value for the EXIT (LOCAL) key is 75. If the EXIT (LOCAL) key is pressed, the script aborts. If any other key is pressed, the script continues.

Digital I/O

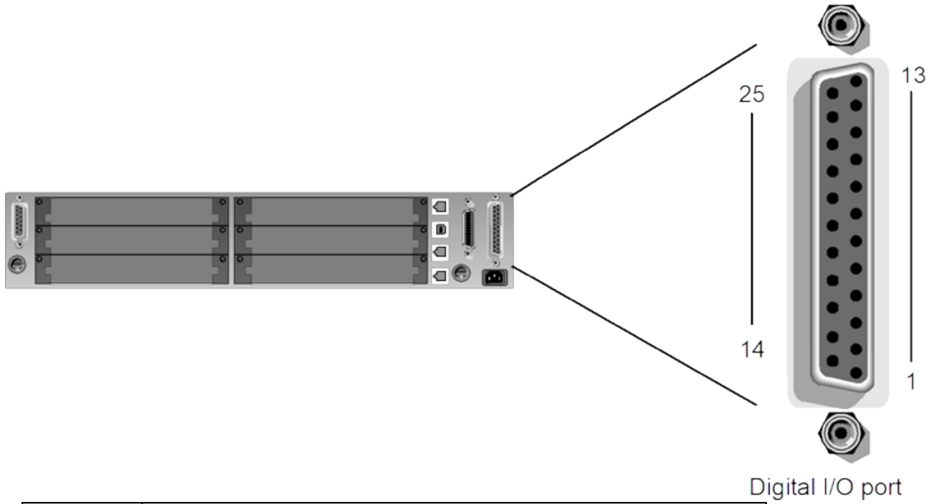
Digital I/O port

The Keithley Instruments Series 3700A System Switch/Multimeter has a digital input/output port that can be used to control external digital circuitry. For example, a handler that is used to perform binning operations can be used with a digital I/O port.

Port configuration

The digital I/O port, a standard female DB-25 connector (shown below), is located on the rear panel.

Figure 59: Digital I/O port



Pin	Description
1 ... 9	Digital I/O #1 ... Digital I/O #9
10 ... 14	Digital I/O #10 (high-current pins; see NOTE) ... Digital I/O #14
15 to 21	Ground
22	V EXT
23	V EXT
24	Pin reserved for future use
25	V EXT

NOTE

For a schematic diagram of the digital I/O hardware, refer to the Series 3700A Specifications on the Keithley Instruments support website (*http://www.keithley.com/support*).
High-current pins (pins 10 to 14) can be used for binning applications or for external relays.

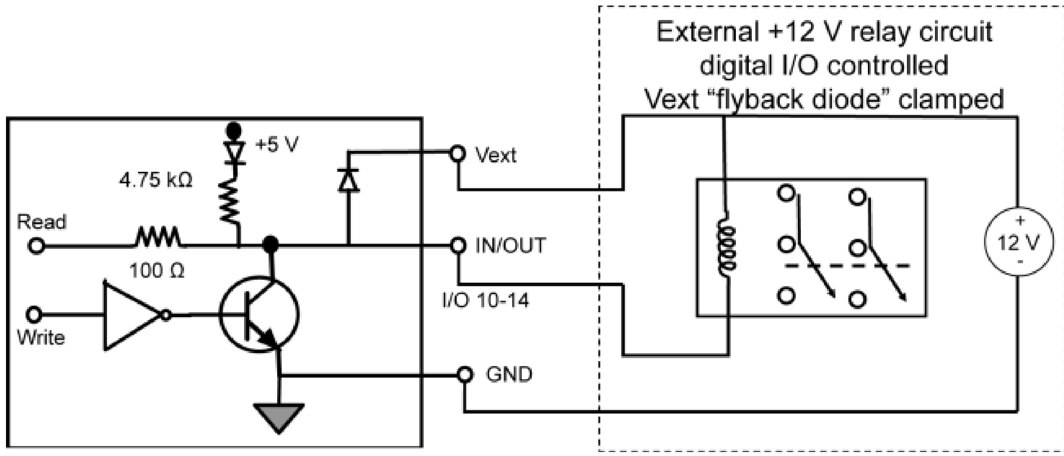
Connecting cables

Use a cable equipped with a standard male DB-25 connector (Keithley Instruments part number CA-126-1).

Vext

The Series 3700A digital I/O provides flyback diode pins named Vext. When connected, Vext can clamp external inductive circuitry (for example, relay drive coils) from +5 V to +33 V. Refer to the figure below for a simplified digital I/O schematic.

Figure 60: Vext flyback diode digital I/O schematic



Hardware interlocks

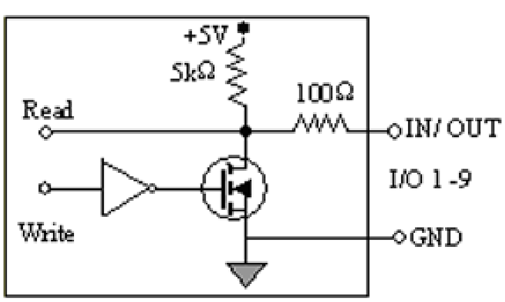
Some switching cards are capable of switching high-voltage signals. For safety reasons, hardware interlocks are provided. The hardware interlocks are present on the switching card itself and are designed to keep the switching card disconnected from the system backplane. This means that when the interlock circuit is disengaged, no measurements can be performed through a switching card. However, channel relays can continue to operate.

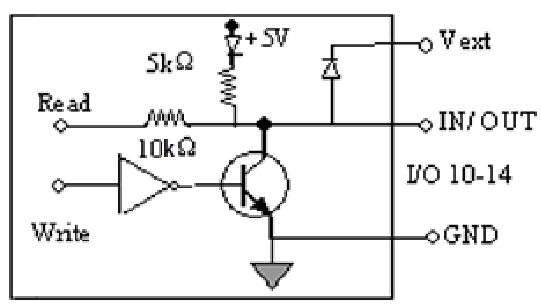
Below is a simplified schematic of the interlock circuit present on the applicable switching cards.

Digital I/O configuration

The following figure shows the basic configuration of the digital I/O port. Writing a 1 to a line sets that line high (~ +5 V). Writing a 0 to a line sets that line low (~0 V). Note that an external device pulls an I/O line low by shorting it to ground, so that a device must be able to sink at least 980 µA per I/O line.

Figure 61: 3700AS Digital I_O port configuration





Controlling digital I/O lines

Although the digital I/O lines are primarily intended for use with a device handler for limit testing, they can also be used for other purposes, such as controlling external logic circuits. You can control lines either from the front panel or over a remote interface.

> **NOTE**
>
> You must write a 1 to all digital I/O lines that are to be used as inputs.
> The trigger mode for the line must be set to `digio.TRIG_BYPASS` in order to use the line for digital I/O. See Trigger model (on page 3-2) for more information.
> The digital I/O lines are not affected by any reset. However, they are affected by a power cycle.

To set digital I/O values from the front panel:

1. Press the **MENU** key, select **DIGIO**, and then press the **ENTER** key or press the navigation wheel ◷.
2. Select **DIGIO-OUTPUT**, and then press the **ENTER** key or the navigation wheel ◷.
3. Set the decimal value as required to set digital I/O lines in the range of 0 to 16,383 (see the table in Digital I/O bit weighting (on page 3-47)), and then press the **ENTER** key or the navigation wheel ◷.
   For example, to set digital I/O lines 3 and 8, set the value to 132.
4. Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To write-protect specific digital I/O lines to prevent their values from being changed:

1. Press the **MENU** key, then select **DIGIO**, and then press the **ENTER** key or the navigation wheel ◷.
2. Select **WRITE-PROTECT**, and then press the **ENTER** key or the navigation wheel ◷.
3. Set the decimal value as required to write-protect digital I/O lines within the range of 0 to 16,383 (see Digital I/O bit weighting (on page 3-47)), and then press the **ENTER** key or the navigation wheel ◷.
   For example, to write-protect digital I/O lines 4 and 10, set the value to 520.
4. Press the **EXIT (LOCAL)** key as needed to return to the main menu.

To remove write protection, reset the decimal value to include only the lines that you want to write protect. To remove write protection from all lines, set the value to 0.

Digital I/O bit weighting

Bit weighting for the digital I/O lines is shown in the following table.

Digital bit weight

Line #	Bit	Decimal weighting	Hexadecimal weighting
1	B1	1	0x0001
2	B2	2	0x0002
3	B3	4	0x0004
4	B4	8	0x0008
5	B5	16	0x0010
6	B6	32	0x0020
7	B7	64	0x0040
8	B8	128	0x0080
9	B9	256	0x0100
10	B10	512	0x0200
11	B11	1,024	0x0400
12	B12	2,048	0x0800
13	B13	4,096	0x1000
14	B14	8,192	0x2000

Remote digital I/O commands

Commands that control and access the digital I/O port are summarized in the following table. See the TSP command reference (on page 8-1) for complete details on these commands. See the following table for decimal and hexadecimal values used to control and access the digital I/O port and individual lines. Use these commands to trigger the Series 3700A using external trigger pulses applied to the digital I/O port, or to provide trigger pulses to external devices.

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

NOTE

The digital I/O lines can be used for both input and output. You must write a 1 to all digital I/O lines that are to be used as inputs.

Remote digital I/O commands

Command	Description
`digio.readbit(`*`bit`*`)`	Read one digital I/O input line
`digio.readport()`	Read digital I/O port
`digio.writebit(`*`bit, data`*`)`	Write data to one digital I/O output line
`digio.writeport(`*`data`*`)`	Write data to digital I/O port
`digio.writeprotect = `*`mask`*	Write protect mask to digital I/O port

Digital I/O programming example

The programming commands below illustrate how to set bit B1 of the digital I/O port high, and then read the entire port value.

```
digio.trigger[1].mode = digio.TRIG_BYPASS
-- Set Bit B1 high.
digio.writebit(1,1)
-- Read digital I/O port.
data = digio.readport()
```

TSP-Link synchronization lines

The Series 3700A has three synchronization lines that you can use for triggering, digital I/O, and to synchronize multiple instruments on a TSP-Link® network.

NOTE

The Models 2604A, 2614A, and 2634A do not have a TSP-Link® interface.

Connecting to the TSP-Link system

The TSP-Link® synchronization lines are built into the TSP-Link connection. Use the TSP-Link connectors located on the back of the Series 3700A. If you are using a TSP-Link network, you do not have to modify any connections. See TSP-Link system expansion interface (on page 7-45) for detailed information about connecting to the TSP-Link system.

Using TSP-Link synchronization lines for digital I/O

Each synchronization line is an open-drain signal. When using the TSP-Link® synchronization lines for digital I/O, any node that sets the programmed line state to zero (0) causes all nodes to read 0 from the line state. This occurs regardless of the programmed line state of any other node. See the table in the Digital I/O bit weighting (on page 3-47) topic for digital bit weight values.

Remote TSP-Link synchronization line commands

Commands that control and access the TSP-Link® synchronization port are summarized in the following table. See the TSP command reference (on page 8-1) for complete details on these commands. See the table in Digital I/O bit weighting (on page 3-47) for the decimal and hexadecimal values used to control and access the digital I/O port and individual lines.

Use the commands in following table to perform basic steady-state digital I/O operations; for example, you can program the Series 3700A to read and write to a specific TSP-Link synchronization line or to the entire port.

NOTE

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

Remote synchronization line commands

Command	Description
`tsplink.readbit(bit)`	Reads one digital I/O input line.
`tsplink.readport()`	Reads the digital I/O port.
`tsplink.writebit(bit, data)`	Writes data to one digital I/O line.
`tsplink.writeport(data)`	Writes data to the digital I/O port.
`tsplink.writeprotect = mask`	Sets write-protect mask of the digital I/O port.

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.trigger[1].mode = tsplink.TRIG_BYPASS
-- Set bit B1 high.
tsplink.writebit(1, 1)
-- Read I/O port.
data = tsplink.readport()
```

Reading buffers

Buffer overview

Reading buffers capture measurements, channels or channel patterns, instrument status, and measure functions of the Keithley Instruments Series 3700A.

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

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

Create and configure buffers using the front panel or through a remote interface using remote commands.

CAUTION

Once you create a reading buffer, if you use that buffer name for another buffer or variable, you can no longer access the original buffer.

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

NOTE

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

Front-panel buffer operation

In the following procedures, pressing in the navigation wheel ◔ will perform the same function as pressing the **ENTER** key. Also, you can turn the navigation wheel ◔ instead of using the **CURSOR** keys.

Read Creating and selecting a reading buffer (on page 3-50) or Selecting a reading buffer (on page 3-51) before performing the following procedures:

- Storing readings (on page 3-51)
- Saving readings (on page 3-51)
- Clearing readings (on page 3-52)
- Deleting a reading buffer (on page 3-52)
- Recalling readings (on page 3-52)
- Buffer configuration (front panel) (on page 3-53)
- Appending readings (on page 3-53)

Creating and selecting a reading buffer

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

1. Press the **STORE** key.
2. Select **CREATE** from the buffer choices and press the **ENTER** key.
3. Using the navigation wheel ◔ and the **CURSOR** keys, scroll through the characters, changing them until the desired name is shown.
   `NAME = _ _ _ _ _ _ _ _ _ _ _ _`
4. Press the **ENTER** key. The starting name is:
   `f p b u f n _ _ _ _ _ _`
   Where:
   `fp` = front panel
   `buf` = buffer
   n = number, sequentially incremented
5. Specify the number of readings to store in the buffer.
6. Press the **ENTER** key. The append attribute of this buffer is enabled (set to 1).

NOTE

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

Selecting a reading buffer

You can only select an existing reading buffer. If necessary, create it first. See Creating and selecting a reading buffer (on page 3-50) for more information.

NOTE

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

To select a reading buffer:

1. Set up the Model 3706A to take measurements.
2. Press the **STORE** key.
3. Select **SELECT** from the buffer choices and press the **ENTER** key.
4. Use the **CURSOR** keys to select the desired buffer.

Storing readings

Before storing readings, make sure you have selected the desired reading buffer. See Selecting a reading buffer (on page 3-51) for more information.

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

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

NOTE

Stored readings are lost when the instrument is turned off. To save your stored readings, see Saving readings (on page 3-51).

Saving readings

When saving readings to a USB flash drive, you must select a non-empty reading buffer. See Selecting a reading buffer (on page 3-51) for more information.

To save readings to a USB flash drive:

1. Select a reading buffer that is not empty.
2. Press the **STORE** key. The bufferVar MENU is displayed.
3. Select the **SAVE** menu item, and press the **ENTER** key. The SAVE RD BUFFER menu is displayed.
4. Press the **ENTER** key when **USB** is highlighted.
5. Using the navigation wheel ◔ and **CURSOR** keys, enter the filename where the data will be saved on the installed USB flash drive. The starting name is:

   `<reading buffer name>_nn_ _ _`

   Where: `nn` starts at `01` and automatically increments. For example, if the selected reading buffer is `fpbuf1`, then the starting name is `fpbuf1_01_ _ _`.
6. Press the **ENTER** key to save the data to the installed USB flash drive or the **EXIT** key to cancel.

Clearing readings

When clearing readings, you must select a reading buffer. See Selecting a reading buffer (on page 3-51) for more information.

To clear readings:

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

Deleting a reading buffer

To delete a reading buffer:

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

NOTE

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

Recalling readings

When recalling readings, you must select a reading buffer that is not empty. See Selecting a reading buffer (on page 3-51) for more information.

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

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

- Measurement reading for each entry is at the top right.
- Buffer location number is at the bottom left.
- Timestamp (on page 3-52) (if used) is positioned at the bottom right.
- Channel display (on page 3-53) or channel pattern (if used) associated with the reading for each entry is at the top left.

Timestamp

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

Channel display

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

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

Displaying other buffer readings and statistics

To display other readings in the reading buffer:

1. While still in the buffer recall mode, if viewing the data stored in the buffer, turn the navigation wheel ◷ to increment and decrement the selected digit of the location number by one. Press the navigation wheel ◷ and then turn it or use the **CURSOR** keys to move to the next digit that the navigation wheel ◷ will change.
2. To exit from the reading buffer recall mode, press the **EXIT (LOCAL)** key.

Buffer configuration (front panel)

When configuring the buffer through the front panel, you must select a reading buffer. See Selecting a reading buffer (on page 3-51) for more information.

To configure a buffer from the front panel:

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

Appending readings

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

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

See bufferVar.appendmode (on page 8-18) for more information.

NOTE

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

Remote buffer operation

You can control the Model 3706A buffer using remote commands.

Data store (buffer) commands

The following commands are associated with data store operation:

- dmm.appendbuffer() (on page 8-152)
- dmm.buffer.catalog() (on page 8-158)
- dmm.buffer.info() (on page 8-159)
- dmm.buffer.maxcapacity (on page 8-160)
- dmm.buffer.usedcapacity (on page 8-160)
- dmm.makebuffer() (on page 8-204)
- dmm.savebuffer() (on page 8-236)

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

Command	Description
`dmm.appendbuffer()`	Creates a file on a USB flash drive if it doesn't already exist. If the file already exists on the flash drive, it will be overwritten.
`dmm.buffer.catalog()`	An iterator that can act on all reading buffer names in the system.
`dmm.buffer.info("buffer name")`	Returns the number of stored readings in the specified buffer, along with the overall buffer capacity. The first returned value is the stored readings number, while the second is the capacity.
`dmm.buffer.maxcapacity`	Returns the overall maximum storage capacity of all reading buffers in the system.
`dmm.buffer.usedcapacity`	Returns the sum storage capacity allocated for all currently created reading buffers in the system.
`dmm.makebuffer()`	Creates buffer to sore measurement data.
`dmm.savebuffer()`	Saves buffer data to a file on a USB flash drive. If the file already exists on the flash drive, it will be overwritten.

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

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

Sample output:

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

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

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

The output is 1.500000000e+004.

This means the system is allocating 15000 of its maximum storage capacity for reading buffers.

Reading buffers

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

Reading buffer designations

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

```
-- Sets how many readings to take with the dmm.measure command.
dmm.measurecount = 5
-- Takes the measurements and stores them in readingbuffer.
dmm.measure(readingbuffer)
```

NOTE

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

Buffer storage control attributes ****3700A***

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

Attribute	Description
appendmode	When off, a new measurement to this buffer will clear the previous contents before storing the new measurement. When on, the first new measurement will be stored at what was formerly rb[n+1]. This attribute is initialized to off when the buffer is created over the bus. However, the default is on for the front panel or web interface to allow triggered readings to fill a buffer without clearing the previous ones.
cachemode	When this attribute is on, the reading buffer cache improves access speed to reading buffer data. When running successive operations that overwrite reading buffer data without running any commands that automatically invalidate the cache, the reading buffer may return stale cache data. This attribute is initialized to on when the buffer is created.
collectchannels	When on, channel or channel pattern information is stored with readings in the buffer. This requires eight extra bytes of storage per reading. This value, off or on, can only be changed when the buffer is empty (cleared). When the buffer is created, this attribute is initialized to on.
collecttimestamps	When on, timestamps will be stored with readings in the buffer. This requires eight extra bytes of storage per reading. This value, off or on, can only be changed when the buffer is empty (cleared). When the buffer is created, this attribute is initialized to on.

Buffer read-only attributes

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

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

Buffer programming examples

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

NOTE

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

Command	Description
`readingbuffer.collectchannels = 1`	Enable channel storage.
`readingbuffer.appendmode = 1`	Enable the buffer append mode.
`readingbuffer.collecttimestamps = 0`	Disable timestamp storage.

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

Command	Description
`number = readingbuffer.n`	Request number of readings stored in the buffer.
`buffer_size = readingbuffer.capacity`	Request the buffer storage capacity.

Buffer reading attributes

The table in Buffer recall attributes (on page 3-57) lists the attributes that control which elements are recalled from the buffer. To access specific elements, append the desired attribute to the buffer designation.

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

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

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

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

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

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

Buffer recall attributes

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

NOTE

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

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

Recall attribute	Description
`statuses`	An array (a Lua table) of status values for all readings in the buffer. The status values are floating-point numbers that encode the status value into a floating-point value (see the table in Buffer status (on page 3-59)).
`times`	An array (a Lua table) of strings, indicating the time of the reading formatted in hours, minutes, and seconds.
`timestamps`	An array (a Lua table) of strings, indicating the timestamp of the reading formatted in month, day, year, hours, minutes, seconds, and fractional seconds.
`fractionalseconds`	An array (a Lua table) of the fractional portion of the timestamps, in seconds, of when each reading occurred. These are absolute fractional times.
`seconds`	An array (a Lua table) of the seconds portion of the timestamp when the reading was stored, in seconds. These seconds are absolute and in UTC format.
`units`	An array (Lua table) of the strings, indicating the unit of measure stored with readings in the buffer. Units may be designated as one of the following: Volts AC, Volts DC, Amps AC, Amps DC, dB VAC, dB VDC, Ohms 2wire, Ohms 4wire, Ohms ComSide, Fahrenheit, Kelvin, Celsius, Hertz, Seconds, and Continuity.

Example to access recall attributes

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

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

Time and date values

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

Time and date values are represented as the number of seconds since some base. Representing time as a number of seconds is referred to as “standard time format.” The three time bases used for the Series 3700A are:

- **UTC 12:00 am Jan 1, 1970.** Some examples of UTC time are reading buffer seconds, adjustment dates, and the value returned by `os.time()`.
- **Instrument on.** References time to when the instrument was turned on. The value returned by `os.clock()` is referenced to the turn-on time.
- **Event.** Time referenced to an event, such as the first reading stored in a reading buffer.

Buffer status

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

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

Buffer status bits

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

Dynamically-allocated buffers

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

Example 1

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

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

Example 2

To delete an allocated buffer named mybuffer:

```
mybuffer = nil
```

Example 3

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

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

Dynamic buffer programming example

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

```
-- Reset the DMM.
dmm.reset("all")
-- Create a buffer named mybuffer and allocate space for 100,000 readings.
mybuffer = dmm.makebuffer(100000)
-- Enable append buffer mode.
mybuffer.appendmode = 1
-- Set count to 1.
dmm.measurecount = 1
-- Select the DMM function as DC volts.
dmm.func = dmm.DC_VOLTS
-- Start for . . . do loop. Measure and store readings in buffer. End loop.
for x = 1, 100 do
      dmm.measure(mybuffer)
end
-- Return readings 1 through 100.
printbuffer(1, 100, mybuffer.readings)
-- Return units 1 through 100.
printbuffer(1, 100, mybuffer.units)
```

Buffer for . . . do loops

The following examples illustrate the use of for . . . do loops when recalling buffer data from a reading buffer called `mybuffer`. The following code may be sent as one command line or as part of a script. Sample outputs follow the line of code. Also see the printbuffer() (on page 8-307) command.

NOTE

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

This example loop uses the `printbuffer()` command to show the reading, units, and relative timestamps for all readings stored in the 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
```

Sample comma-delimited output of above code:

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

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

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

Sample columnar output of above code:

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

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

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

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

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

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

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

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

Exceeding reading buffer capacity

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

Example

Create a buffer with:

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

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

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

The code for the previous example follows:

```
--  Create a buffer named buf and allocate space for 50 readings.
buf = dmm.makebuffer(50)
--  Enable append buffer mode.
buf.appendmode = 1
--  Set count to 30.
dmm.measurecount = 30
--  Show the current number of readings in the buffer,
-- and then measure and store readings in the
-- buffer (first pass).
--  Output from the print command:
--  0.000000000e+000
--  5.245720223e-002
print(buf.n, dmm.measure(buf))
--  Show the current number of readings in the buffer,
-- and then measure and store readings in the
-- buffer (second pass).
--  Output from the print command:
--  3.000000000e+001
--  -1.388141960e-001
--  4915, Attempting to store past capacity of reading buffer
print(buf.n, dmm.measure(buf))
--  Show the current number of readings in the buffer,
-- and then measure and store readings in the
-- buffer (third pass).
--  Output from the print command:
--  5.000000000e+001
--  nil
--  4915, Attempting to store past capacity of reading buffer
print(buf.n, dmm.measure(buf))
```

Basic DMM operation

In this section:

DMM measurement capabilities

NOTE

The DMM is not available on the Models 3706-S or 3706-SNFP.

The DMM of the Model 3706A can make the following measurements:

- DC voltage measurements from –10 nV to 300 V
- AC voltage measurements from 100 nV to 300 V
- DC current measurements from 1pA to 3.0 A
- AC current measurements from 1 nA to 3.0 A
- Two-wire resistance measurements from 1 μΩ to 120 MΩ
- Four-wire resistance measurements from 100 μΩ to 120 MΩ
- Commonside ohms resistance measurements from 1 μΩ to 2 kΩ
- Frequency measurements from 3 Hz to 500 kHz
- Period measurements from 2 μs to 330 ms
- Temperature measurements from –200 °C to 1820 °C
- Continuity testing using the 1 kΩ range

⚠ CAUTION

When using a switching module, do not exceed the maximum signal levels of the module.

NOTE

For information on verification and calibration of the DMM, see Verification (on page C-1).

High-energy circuit safety precautions

To optimize safety when measuring voltage in high-energy distribution circuits, read and use the directions in the following warning.

WARNING

Dangerous arcs of an explosive nature in a high-energy circuit can cause severe personal injury or death. If the multimeter is connected to a high-energy circuit when set to a current range or low resistance range, the circuit is virtually shorted. Dangerous arcing can result even when the multimeter is set to a voltage range if the minimum voltage spacing is reduced in the external connections.

As described in International Electrotechnical Commission (IEC) Standard IEC 664, the Model 3706A is Installation Category I and signal lines must not be directly connected to AC mains.

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.

Use the following procedure when testing power circuits:

1. Turn off power to the circuit using the regular 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 multimeter 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.

Performance considerations

There are several settings and conditions that apply to all DMM measurements, as described in the following sections.

Warmup time

After the Model 3706A is turned on, it must be allowed to warm up for at least two hours to allow the internal temperature to stabilize. If the instrument has been exposed to extreme temperatures, allow extra warmup time.

Autozero

When the autozero feature is enabled, the Model 3706A periodically measures internal voltages that correspond to offset (zero) and amplifier gain reference points. The Model 3706A includes these measurements when it calculates the reading of the input signal. This helps maintain stability and accuracy over time and changes in temperature.

You can disable autozero to improve measurement speed. However, if autozero is disabled for long periods, the zero and gain reference points will drift, resulting in inaccurate readings.

NOTE

To maintain accuracy of your DMM readings, you should disable autozero for only short periods.

When autozero is enabled after being disabled for a long period, the internal reference points are not updated immediately. This initially results in inaccurate measurements, especially if the ambient temperature has changed by several degrees.

To force a rapid update of the internal reference points, you can set the AUTOZERO attribute for the function to ONCE. This updates the internal reference points once and stops. See dmm.autozero (on page 8-157) for more information. For example, you could send the following commands to update the internal reference points and then enable autozero if autozero had been set to off for a long period:

```
dmm.autozero = dmm.AUTOZERO_ONCE
dmm.autozero = dmm.ON
first_reading = dmm.measure()
```

NOTE

Internal temperature references used for thermocouple measurements are performed regardless of the autozero state because they do not have a significant effect on measurement speed.

To set autozero from the front panel for the selected function:

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Use the navigation wheel ◷ to select AUTOZERO.
4. Select:
   - OFF to disable autozero
   - ON to enable autozero
   - ONCE to update the reference points once and then disable autozero
1. Press the navigation wheel ◷ or **ENTER** to save the setting for the selected function.

To set autozero remotely, see dmm.autozero (on page 8-157).

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 DMM functions:

- Commonside ohms
- Continuity
- DC current
- DC volts
- Four wire ohms
- Temperature
- Two wire ohms

Line synchronization can be enabled for NPLC measurements, increasing NMRR and CMRR.

To set line synchronization from the front panel for the selected function:

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Use the navigation wheel to select LINESYNC.
4. Select:
   - OFF to disable line synchronization
   - ON to enable line synchronization
1. Press the navigation wheel or **ENTER** to save the setting for the selected function.

To set line synchronization remotely, see dmm.linesync (on page 8-203).

Autodelay

Autodelay applies a wait period at the start of measurement. The delay allows cables, Series 3700A cards, or internal DMM circuitry to settle for best measurement accuracy. For the AC current and AC volts functions, the autodelay includes both the RMS filter and AC coupling capacitor settling times.

When autodelay is disabled, no wait time is applied. When autodelay is enabled, every start of measurement for a function is delayed by the same amount of time.

You can also use autodelay once to include a delay for only the first measurement in a set of measurements. Each measurement after the first one has no additional delay.

To set autodelay from the front panel for the selected function:

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Use the navigation wheel ◷ to select **AUTODELAY**.
4. Select:
   - **OFF** to disable auto delay
   - **ON** to enable auto delay
   - **ONCE** to enable auto delay for only the first measurement
1. Press the navigation wheel ◷ or **ENTER** to save the setting for the selected function.

To set autodelay remotely, dmm.autodelay (on page 8-154).

Measure count

The DMM can be set up to take multiple measurements when **MEASURE** is selected from the DMM Action Menu on the front panel or when a single trigger is sent. This is useful in channel closures or in a scan list where multiple measurements are required per channel.

To set up multiple measurements from the front panel:

1. Press the **DMM** key.
2. Use the navigation wheel ◷ to select COUNT.
3. Set the number of measurements to take (maximum of 450,000).
4. Press the navigation wheel ◷ or **ENTER** to save the setting. Note that this settings affects all measurements for all functions (it is not tied to a specific function).

To set the measurement counts remotely,see dmm.measurecount (on page 8-214).

When continuous measurements are taken with the front panel TRIG key, they are taken at 250 ms intervals. You take continuous measurements by pressing and holding the TRIG key in for a few seconds. After doing this, the TRIG annunciator will flash to indicate that readings are being triggered.

The front panel TRIG key will perform the number of measurements equal to the measurecount number. Press EXIT or send `dmm.measurecount=1` to halt triggering.

Change the display resolution

You can set the display resolution for measurements that are shown on the front panel of the instrument. You can set the resolution to 3½, 4½, 5½, 6½, or 7½ digits. Display resolution can be set for all functions except "nofunction" and "continuity".

The display resolution does not affect the number of digits returned in a remote command reading, and does not affect the accuracy or speed of measurements.

To set the display resolution delay from the front panel for the selected function:

1. Press the **CONFIG** key.
2. Press the **DMM** key.
3. Use the navigation wheel to select DIGITS.
4. Select 3, 4, 5, 6,or 7 to select a 3½, 4½, 5½, 6½, or 7½ digit display, respectively.
5. Press the navigation wheel or **ENTER** to save the setting for the selected function.

To set display resolution remotely, see dmm.displaydigits (on page 8-180).

System considerations

Relationship between DMM functions and attributes

Each DMM function can be modified by a set of attributes. For example, you can use the relative offset attribute to set a value that zeroes out noise in a measurement.

Attribute settings apply only to the function that is active when the attribute is set. They remains in effect for that function until the instrument is powered off, reset, or a saved configuration is recalled.

If you set the same attribute for a different function, the setting changes for the new function, but does not change for the previous function. An example is shown here using the remote commands, but the same concept applies to front panel settings.

`reset()` `dmm.func = "acvolts"` `print(dmm.func, dmm.autorange)`	Reset the instrument. Set the function to AC volts. Print the function and autorange values. Output: `acvolts      1.000000000e+000` This indicates that the active function is AC volts, with autorange set on.
`dmm.func = "accurrent"` `dmm.autorange = "dmm.OFF"` `print(dmm.func, dmm.autorange)`	Change to the AC current function and turn autorange off. Print the function and autorange values. Output: `accurrent    0.000000000e+000` This indicates that the active function is AC current, with autorange set off.
`dmm.func = "acvolts"` `print(dmm.func, dmm.autorange)`	Return to AC volts as the active function. Print the function and autorange values. Output: `acvolts      1.000000000e+000` Note that when AC volts is the active function, the autorange value is on.

Relationship between front panel settings and remote commands

When you change the active DMM function from the front panel, the active function for all other interfaces is changed as well. This is also true for attributes.

When you change the active DMM function through a remote interface, the front panel settings are also changed.

Save DMM configurations

After you set up the attributes for a DMM function, you can save the settings for future use using the DMM configuration options. You can use this configuration to:

- Switch quickly between set ups
- Assign the configuration to channels or channel patterns
- Assign the configuration to scanning

DMM configurations include all the attribute settings for the selected DMM function. You can save multiple configurations with different names so that you can save different sets of attributes for the DMM functions.

When the DMM configuration is assigned to channels, channel patterns or scans, the Model 3706A verifies that the attributes that are set in the DMM configuration are valid for all the channels included in the channel list, pattern, or scan. If there are attributes that are not valid, an error occurs and you cannot take measurements.

However, the DMM configuration and the pole setting of the channel can be incompatible without causing an error. For example, the DMM configuration may have a channel configured for two-pole measurement (for example, DC volts), while the pole setting may be configured for four-pole. Or, a channel may have a DMM configuration of four-wire ohms while the pole setting is at two-pole. If the pole settings conflict, the pole settings that take precedence depend on the operation:

- If you send `dmm.close` or `dmm.open`, the pole setting in the DMM configuration is used.
- If you send `channel.close`, `channel.open`, `channel.exclusiveclose`, or `channel.exclusiveslotclose`, the pole setting of the channel is used.

The DMM configurations are saved in configuration scripts. See Create a configuration script (on page 2-102) for more information on configuration scripts.

DMM configurations are deleted if the system is reset. They are not affected by a DMM reset (`dmm.reset`).

Memory available for DMM configurations

All DMM configurations are allocated 32 KB of memory. The number of DMM configurations you can store varies with the number of characters of the name of the DMM configuration and the number of attributes associated with a particular function. For example, if each DMM configuration name is six characters, you can store 78 temperature configurations (temperature has 41 unique DMM attribute settings). However, if the the function is set to DC volts, and each name is six characters, you can store 99 DMM configurations (DC volts only has 31 unique DMM attribute settings). You can use the DMM configuration query command to determine how many attributes are associated with a function (see dmm.configure.query (see "dmm.configure.query()" on page 8-171)).

To see how much of the DMM configuration memory is available or used, see the memory.available() (on page 8-302) or memory.used() (on page 8-303) commands.

How to save a DMM configuration

To create and save a DMM configuration from the front panel:

1. Press **DMM**.
2. Use the navigation wheel ◷ to select **SAVE**.
3. Press the navigation wheel ◷ or press **ENTER**.
4. Assign a name to the configuration.
5. Press **ENTER**.

To create a configuration script from the web interface:

1. From the Cards listing, select the slot that you want to set up configuration for.
2. Click **DMM Config**. The DMM Configuration dialog box is displayed.
3. Complete the values in the Configuration dialog box as needed.
4. Click **Save as** to save the settings.

To create a configuration script from the remote interface:

Send the command:

```
dmm.configure.set(name)
```

Where `name` is the name you want to assign to the DMM configuration.

Open and close relay operation

The **OPEN** and **CLOSE** keys operate differently if they are configured for switch operation or DMM operation.

If they are configured for DMM operation, backplane relays are automatically closed to connect the channel to the DMM to make a measurement. If you assign a DMM configuration to a channel, the **OPEN** or **CLOSE** keys behave as a DMM operation — the input signals are automatically routed to the DMM through the backplane relays.

In switch operation, backplane relays are not closed unless they are assigned to a channel. When that channel is closed, the associated backplane channel also closes. A backplane relay cannot be closed as a stand-alone channel from the front panel. By default, the channels are not assigned a measurement function (`"nofunction"`) and the **OPEN** or **CLOSE** keys behave as a switch.

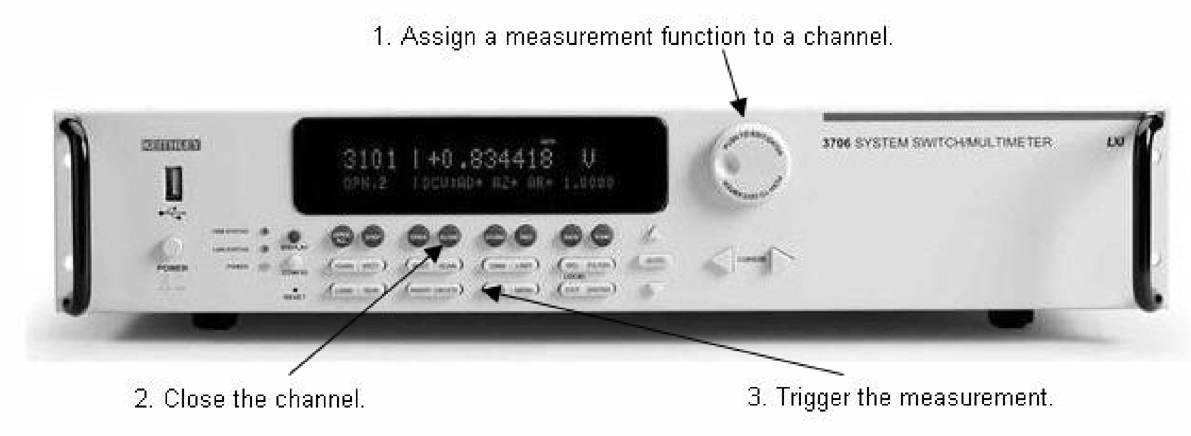
If error code 1114, "Settings conflict error," is displayed, the channel that is being closed has `"nofunction"` assigned to it. For remote operation, to use `dmm.close`, you must assign a valid function to a channel.

Front panel close and open operation is shown in the following example.

Example 1: Close channel and take measurement using the DMM operation method

When you assign a measurement function to a channel and press the **Close** key, the **Close** key routes the input signal automatically to the DMM through the appropriate backplane relays. This behavior is referred to as DMM operation.

Figure 62: Close channel and trigger measurement on instrument using the DMM method



1. Assign a measurement function of two-wire ohms to channel 1031:
    a. Use navigation wheel to modify the channel designation from **001** to **031**.
    b. Use navigation wheel to select **twowireohms** as the measurement function.
1. Press the **CLOSE** key to close channel 1031.
2. Press the **TRIG** key to acquire and display a single measurement.

Voltage measurements (DC volts and AC volts)

The Model 3706A can make DC volt measurements from –10 nV to 300 V and AC volt measurements from 100 nV to 300 V_{RMS} (425 V peak for AC waveforms).

- DC volt input resistance: 100 mV through 10 V ranges: more than 10 GΩ
- 100 V and 300 V ranges: more than or equal to 10 MΩ

Refer to the specifications for complete and up-to-date information and tolerances.

Settings available for voltage measurement

The following DMM attributes are available for voltage measurements:

- aperture (range of 10e-6 s to 0.250 s for 50 Hz; 8.33e-6 s to 0.250 s for 60 Hz)
- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- autorange (`dmm.ON` or `dmm.OFF`)
- autozero (`dmm.AUTOZERO_ONCE`, `dmm.ON` or `dmm.OFF`)
- DB reference (1e-7 V to 1000 V)
- detector bandwidth (3, 30, or 300)
- display digits (3, 4, 5, 6, or 7)
- filter count (1 to 100)
- filter enable (`dmm.ON` or `dmm.OFF`)
- filter type (`dmm.FILTER_MOVING_AVG` or `dmm.FILTER_REPEAT_AVG`)
- filter window (0 to 10%)
- input divider (`dmm.ON` or `dmm.OFF`)
- DMM limit auto clear (`dmm.ON` or `dmm.OFF`)
- DMM limit enable (`dmm.ON` or `dmm.OFF`)
- DMM limit high fail (0 or 1)
- DMM limit high value (–4294967295 to +4294967295)
- DMM limit low fail (0 or 1)
- DMM limit low value (–4294967295 to +4294967295)
- line synchronization (`dmm.ON` or `dmm.OFF`)
- math enable (`dmm.ON` or `dmm.OFF`)
- math format (`dmm.MATH_NONE`, `dmm.MATH_MXB`, `dmm.MATH_PERCENT`, or `dmm.MATH_RECIPROCAL`)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- nplc (0.0005 to 15 for 60 Hz; 0.0005 to 12 for 50 Hz)
- range (0 to 303)
- relative offset enable (`dmm.ON` or `dmm.OFF`)
- relative offset level
- units (`dmm.UNITS_VOLTS` or `dmm.UNITS_DECIBELS`)

Autodelay and auto range settings

The following table provides times for autodelay and autorange time for the Model 3706A DMM functions.

Function	Detector bandwidth	Range and delays					
DC volts	Not applicable	Range	100 mV	1 V	10 V	100 V	300 V
		Autodelay	1 ms	1 ms	1 ms	5 ms	5 ms
		Autorange	1 ms	1 ms	1 ms	5 ms	5 ms
AC volts	Not applicable	Range	100 mV	1 V	10 V	100 V	300 V
	3 or 30 Hz	Autodelay	200 ms	200 ms	200 ms	200 ms	1 s
		Autorange	200 ms	200 ms	200 ms	200 ms	1 s
	300 Hz	Autodelay	50 ms	50 ms	50 ms	50 ms	250 ms
		Autorange	50 ms	50 ms	50 ms	50 ms	250 ms

Voltage measurement connections

WARNING

Even though the Model 3706A can measure up to 300V, the maximum input to a switching module may be less. Exceeding the voltage rating of a switching module may cause damage and create a safety hazard.

Make sure the insulation and wire sizes used are appropriate for the voltages and current being applied to the Model 3706A analog backplane connector. Use supplementary insulation as needed. Exceeding the voltage rating of a wiring may cause damage and create a safety hazard.

Figure 63: DCV connection

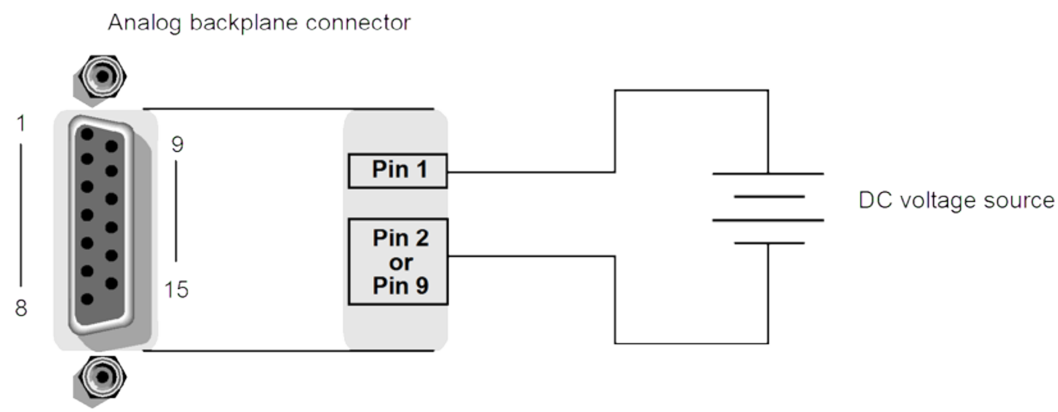
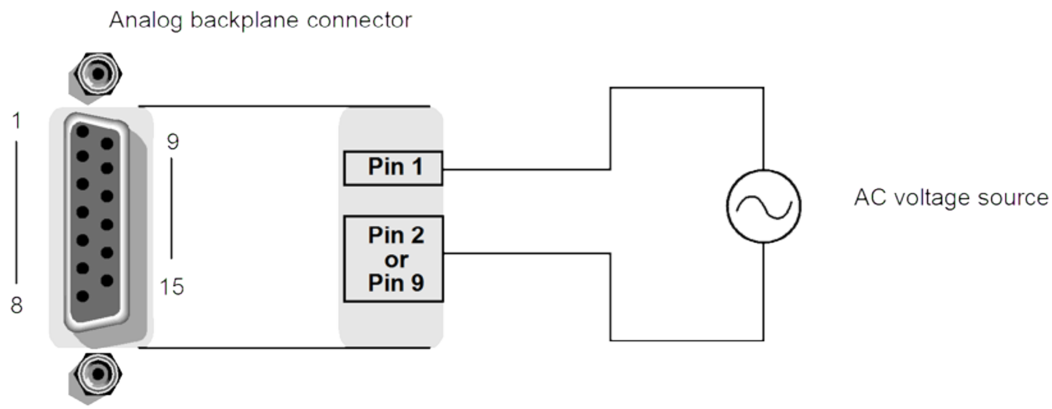


Figure 64: ACV connection



Voltage measurement procedure front panel

⚠ WARNING

If both the analog backplane connector and a switching module's terminals are connected at the same time, all wiring and connections must be rated to the highest voltage that is connected. For example, if 300 V is connected to the analog backplane connector, the test lead insulation for the switching module must also be rated for 300 V.

⚠ CAUTION

Do not apply more than maximum input levels indicated or instrument damage may occur. The voltage limit is subject to the 8×10^7 VHz product.

Perform the following steps to change a DMM function and its attributes.

1. Press the **OPENALL** key to open all switching channels.
2. Select the voltage measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **Function?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys or the knob to select DCV.
3. Use the **RANGE** ⮝ and ⮟ keys to:
   - Select a measurement range
   - Adjust the attributes after selecting the desired function under the Config DMM menu
   - Press the **AUTO** key to select autoranging (AUTO annunciator turns on)

1. Apply the voltages to be measured.
2. If using a Series 3700A switch card, perform the following steps to assign a range of channels and assign the channel a DMM configuration:
   a. Using the navigation wheel:
   - Press once to select 3700 card slot number and adjust 1-6.
   - Press a second time to select the start channel number.
   - Press a third time to select the end channel.
   - Press a fourth time to allow DMM configuration assignment to the channel or range of channels.

   b. Press the **CLOSE** key.

1. Press the **TRIG** key and observe the display. If the "Overflow" message is displayed, select a higher range until a normal reading is displayed (or press the **AUTO** key for autoranging). For manual ranging, use the lowest possible range for the best resolution.
2. To measure other switching channels, repeat steps 5 and 6.
3. When finished, press the **OPENALL** key to open all channels.

Voltage measurement procedure remote commands

WARNING

If both the analog backplane connector and a switching module's terminals are connected at the same time, all wiring and connections must be rated to the highest voltage that is connected. For example, if 300 V is connected to the analog backplane connector, the test lead insulation for the switching module must also be rated for 300 V.

CAUTION

Do not apply more than maximum input levels indicated or instrument damage may occur. The voltage limit is subject to the 8×10^7 V Hz product.

Use `dmm.func` to set the active function to AC volts, then set attributes as needed for your application. An example is shown in the following table.

Example program code for voltage measurement

Code	Notes and comments
`reset()`	Reset the Series 3700A to the factory defaults.
`dmm.func="acvolts"` `dmm.range=1` `dmm.detectorbandwidth=300`	Sets the DMM function to AC volts, with a range of 10 and detector bandwidth of 300.
`dmm.nplc=0.06` `dmm.autozero=0`	When bandwidth set to 300, NPLC can be programmed from 0.0005 plc to 12 plc at 60 Hz or 15 plc at 50 Hz.
`dmm.autodelay=dmm.AUTODELAY_ONCE`	Include a single 50 ms delay before each measurement after channel closure.
`scan.measurecount=25`	DMM takes a 25 readings on the same channel.
`dmm.configure.set("my-1Vac")`	Define this group of DMM settings as "my-1Vac".
`dmm.setconfig("4004, 4024", "my-1Vac")`	Assign the configuration for channels 4 and 24 to "my-1Vac".
`buf=dmm.makebuffer(200)` `buf.clear()` `buf.appendmode=1`	Set the buffer size set to 200 readings, clear the buffer, and set the readings to be appended to the existing buffer content.
`scan.create("4004, 4024")`	Create a scan list that includes channels 4 and 24. Backplane channels 4911 and 4921 are automatically paired.
`scan.scancount=4`	Set the scan to loop 4 times.
`scan.execute(buf)` `for x=1,buf.n do printbuffer(x,x,buf,` `    buf.relativetimestamps)` `end`	Start the scan. Note that `x,x` prints reading and time vertically so you can copy and paste the information into Microsoft® Excel®.

Current measurements (DC current and AC current)

The Model 3706A can make DC current measurements from 1 pA to 3 A and AC current measurements from 1 mA to 3 A_{RMS}.

⚠ WARNING

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

NOTE

Also see crest factor information contained in AC voltage measurements and crest factor (on page 5-5).

Settings available for current measurements

The following DMM attributes are available for AC current and DC current measurements:

- aperture (range of 10e-6 s to 0.250 s for 50 Hz; 8.33e-6 s to 0.250 s for 60 Hz)
- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- autorange (`dmm.ON` or `dmm.OFF`)
- autozero (`dmm.AUTOZERO_ONCE`, `dmm.ON` or `dmm.OFF`)
- display digits (3, 4, 5, 6, or 7)
- filter count (1 to 100)
- filter enable (`dmm.ON` or `dmm.OFF`)
- filter type (`dmm.FILTER_MOVING_AVG` or `dmm.FILTER_REPEAT_AVG`)
- filter window (0 to 10%)
- DMM limit auto clear (`dmm.ON` or `dmm.OFF`)
- DMM limit enable (`dmm.ON` or `dmm.OFF`)
- DMM limit high fail (0 or 1)
- DMM limit high value (–4294967295 to +4294967295)
- DMM limit low fail (0 or 1)
- DMM limit low value (–4294967295 to +4294967295)
- line synchronization (`dmm.ON` or `dmm.OFF`) (dc only)
- math enable (`dmm.ON` or `dmm.OFF`)
- math format (`dmm.MATH_NONE`, `dmm.MATH_MXB`, `dmm.MATH_PERCENT`, or `dmm.MATH_RECIPROCAL`)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- nplc (0.0005 to 15 for 60 Hz; 0.0005 to 12 for 50 Hz)
- range (0 to 3.1)
- relative offset enable (`dmm.ON` or `dmm.OFF`)
- relative offset level

Autodelay and auto range settings

The following table provides times for autodelay and autorange time for the Model 3706A DMM current function.

Function	Detector bandwidth	Range and delays							
DC current	Not applicable	Range	10 µA	100 µA	1 mA	10 mA	100 mA	1 A	3 A
		Autodelay	13 ms	2 ms	2 ms	2 ms	2 ms	2 ms	2 ms
		Autorange	13 ms	2 ms	2 ms	2 ms	2 ms	2 ms	2 ms
AC current	Not applicable	Range			1 mA	10 mA	100 mA	1 A	3 A
	3 or 30 Hz	Autodelay			200 ms	200 ms	200 ms	200 ms	300 ms
		Autorange			200 ms	200 ms	200 ms	200 ms	300 ms
	300 Hz	Autodelay			50 ms	50 ms	50 ms	50 ms	75 ms
		Autorange			50 ms	50 ms	50 ms	50 ms	75 ms

Current measurement connections

See the Model 3721 information in the Series 3700 Switch and Control Cards Reference Manual for connection information.

NOTE

The Model 3721 switch card is the only card that supports DC current and AC current functions. You can only assign DC current or AC current to channels 41 and 42. If DC current or AC current is assigned to Channel 1-40, error 1116 "function mismatch in configuration" is displayed. Also, if a function other than DC or AC current is assigned to channel 41 or 42, error 1114 "function mismatch in configuration" is displayed.

Current measurement procedure from the front panel

1. Press the **OPENALL** key to open all switching channels.
2. Select the current measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **Function?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys or the knob to select ACI or DCI.
3. Use the **RANGE** ⋀ and ⋁ keys to:
   - Select a measurement range
   - Adjust the attributes after selecting the desired function under the Config DMM menu

1. Press the **AUTO** key to select autoranging (AUTO annunciator turns on).
2. Apply the currents to be measured.
3. If using a Series 3700A switch card, perform the following steps to assign a range of channels and assign the channel a DMM configuration:
   a. Using the navigation wheel:
   - Press once to select 3721 card slot number and adjust 1-6.
   - Press a second time to select the start channel number.
   - Press a third time to select the end channel.
   - Press a fourth time to allow DMM configuration assignment to the channel or range of channels.

   b. Press the **CLOSE** key.

1. Press the **TRIG** key and observe the display. If the "Overflow" message is displayed, select a higher range until a normal reading is displayed (or press the **AUTO** key for autoranging). For manual ranging, use the lowest possible range for the best resolution.
2. To measure other switching channels, repeat steps.
3. When finished, press the **OPENALL** key to open all channels.

NOTE

When an amps-only channel is closed, you cannot select a non-amps function.

When making measurements less than 1 µA, to minimize 50/60 Hz noise, use a twisted pair for current and DMM connections.

Current measurement procedure through remote commands

To set the DMM function for AC current measurements, send the command:

```
dmm.func = "accurrent"
```

To set the DMM function for DC current measurements, send the command:

```
dmm.func = "dccurrent"
```

Resistance measurements

The Model 3706A can make resistance measurements from 0.1 µΩ to 120 MΩ. For resistances more than 1 kΩ, the two-wire method is typically used for measurements. For resistances more than or equal to 1 kΩ, the four-wire measurement method should be used to cancel the effect of test lead and channel path resistances.

DMM resistance measurement methods

The method that the Model 3706A uses to measure resistance depends on the resistance range. For resistance ranges from 1 Ω to 1 MΩ , the Model 3706A uses the constant-current method to measure resistance. For resistance ranges from 10 MΩ to 100 MΩ ranges, the ratiometric method is used.

When the constant-current method is used, the Model 3706A 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 6.4 V reference through a 10 MΩ reference resistance (R_{REF}).

For more detail on these methods, see Constant-current source method (on page 5-9) and Ratiometric method (on page 5-9).

The Model 3706A uses four methods to detect open leads. For detail, see Open lead detection (on page 5-14).

Settings available for resistance measurements

The following DMM attributes are available for resistance measurements:

- aperture (range of 10e-6 s to 0.250 s for 50 Hz; 8.33e-6 s to 0.250 s for 60 Hz)
- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- autorange (`dmm.ON` or `dmm.OFF`)
- autozero (`dmm.AUTOZERO_ONCE`, `dmm.ON` or `dmm.OFF`)
- display digits (3, 4, 5, 6, or 7)
- dry circuit (`dmm.ON` or `dmm.OFF`) (only for four-wire ohms and commonside ohms)
- filter count (1 to 100)
- filter enable (`dmm.ON` or `dmm.OFF`)
- filter type (`dmm.FILTER_MOVING_AVG` or `dmm.FILTER_REPEAT_AVG`)
- filter window (0 to 10%)
- DMM limit auto clear (`dmm.ON` or `dmm.OFF`)
- DMM limit enable (`dmm.ON` or `dmm.OFF`)
- DMM limit high fail (0 or 1)
- DMM limit high value (–4294967295 to +4294967295)
- DMM limit low fail (0 or 1)
- DMM limit low value (–4294967295 to +4294967295)
- line synchronization (`dmm.ON` or `dmm.OFF`)
- math enable (`dmm.ON` or `dmm.OFF`)
- math format (`dmm.MATH_NONE`, `dmm.MATH_MXB`, `dmm.MATH_PERCENT`, or `dmm.MATH_RECIPROCAL`)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- nplc (0.0005 to 15 for 60 Hz; 0.0005 to 12 for 50 Hz)
- offset compensation (`dmm.ON` or `dmm.OFF`) (only for four-wire ohms and commonside ohms)
- open detector (`dmm.ON` or `dmm.OFF`) (only for four-wire ohms and commonside ohms)
- range (0 to 120e6)
- relative offset enable (`dmm.ON` or `dmm.OFF`)
- relative offset level

Autodelay and auto range settings

The following table provides times for autodelay and autorange time for the Model 3706A DMM functions.

When measuring resistances that are more than 10 KΩ, cable and Series 3700A card capacitance, along with dielectric absorption, can cause uncertainties, such as low readings. The low readings are caused by insufficient settling time after the closure of a Series 3700A switch card channel. Automatic delays have been optimized to allow proper settling after the close of a channel. See below settling times. If the application requires an additional settling delay, use the following commands to add delays to the channel or slot:

```
channel.setdelay("4004", 0.050)
```

Adds 50 ms of delay after closing channel 4 in slot 4.

```
channel.setdelay("slot4", 0.050)
```

Adds 50 ms of delay to all channels in slot 4.

For continuity, the range is 1 kΩ with an autodelay of 3 ms and auto range of 2.5 ms.

Function	**Range and delays**							
2-wire ohm and 4-wire ohm	**Range**	1 - 100 Ω	1 kΩ	10 kΩ	100 kΩ	1 MΩ	10 MΩ	100 MΩ
	Autodelay	3 ms	3 ms	13 ms	25 ms	100 ms	250 ms	375 ms
	Autorange	2.5 ms	2.5 ms	12.5 ms	25 ms	100 ms	250 ms	375 ms
Dry circuit ohms	**Range**	1 - 10Ω	100 - 2kΩ					
	Autodelay	3 ms	13 ms					
	Autorange	2.5 ms	12.5 ms					

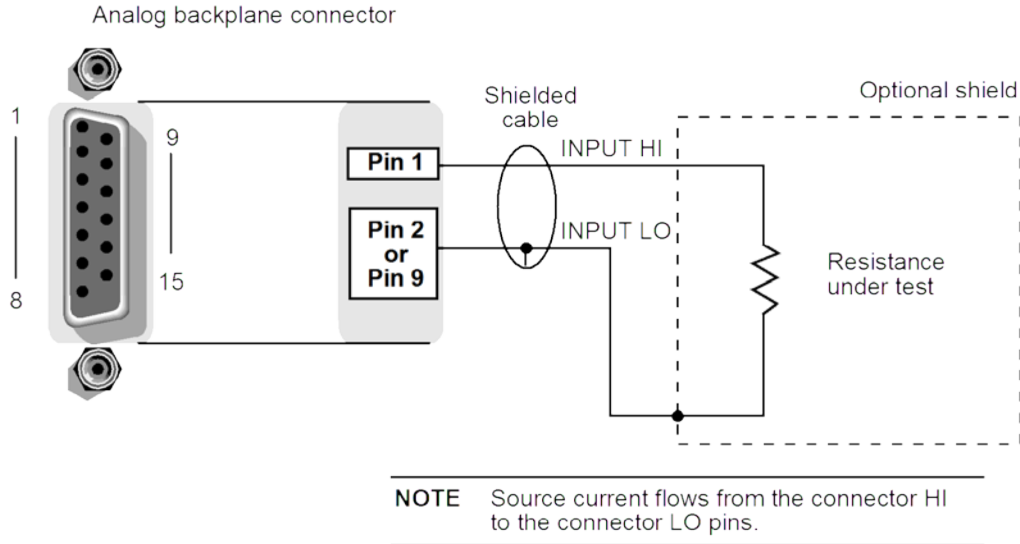
Resistance measurement connections

Analog backplane connector (rear panel)

Connections for resistance measurements are shown below.

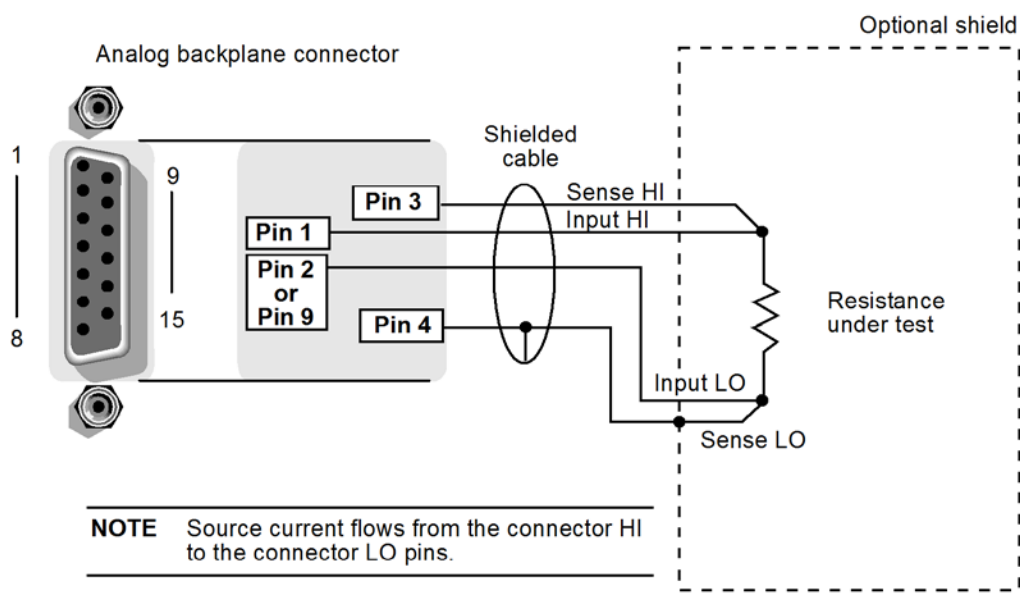
For 2-wire resistance measurements, connect the leads to INPUT HI and LO.

Figure 65: Two-wire resistance measurements



For 4-wire resistance, connect the leads to INPUT HI and LO, and sense Ω4 HI and LO.

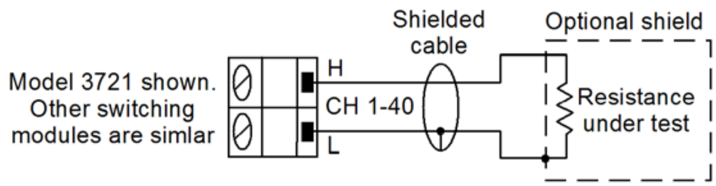
Figure 66: Four-wire resistance measurement



Switching module connections

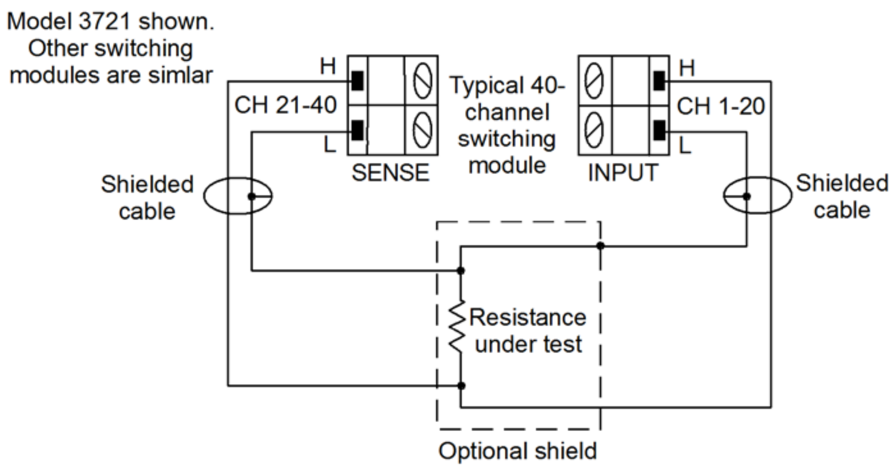
Connections for the switching module are shown below. As shown, each of the 40 channels can be used to perform 2-wire resistance measurements.

Figure 67: Two-wire switching module resistance connection



For 4-wire resistance measurements, a channel pair is used for each 4-wire measurement, as shown below. For 4-wire resistance connections on a 40-channel switching module, channels 1 through 20 (which are used as the INPUT terminals) are paired to channels 21 through 40 (which are used as the SENSE terminals). Channel 1 is paired to channel 21, channel 2 is paired to channel 22, and so on.

Figure 68: Four-wire switching module resistance connection



NOTE	Source current flows from input high (H) to input low (L).

Cable leakage

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

Shielding

To achieve a stable reading, it helps to shield resistances greater than 100 kΩ. As shown in Analog backplane connector (rear panel) (on page 4-19), place the resistance in a shielded enclosure and connect the shield to the INPUT LO terminal of the instrument electrically.

Commonside ohms

The following figure provides a switching schematic for the Model 3721 when measuring 4-wire commonside ohms.

Resistance measurements from the front panel

CAUTION

Inputs: Do not apply more than 425 V peak between INPUT HI and LO. Failure to observe this caution may result in instrument damage.
Switching cards: Do not apply more than 300 V DC or 300 V_{RMS} (425 V_{peak}) for AC waveforms between any two pins. Failure to observe this caution may result in switching module damage.
For example, if INPUT channel 1 HI is 300 VDC from channel 1 LO, channel 1 LO must be ≈ 0 VDC from chassis ground.

Perform the following steps to measure resistance:

1. Press the **OPENALL** key to open all switching channels. Refer to DCV for DMM function, range, and other settings.
2. Connect the resistances to be measured.
3. Select the resistance measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **Function?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys or the knob to select TWOWIREOHMS, FOURWIREOHMS, or COMMONSIDE.
4. Use the **RANGE** ⋀ and ⋁ keys to:
   - Select a measurement range
   - Adjust the attributes after selecting the desired function under the Config DMM menu
   - Press the **AUTO** key to select autoranging (AUTO annunciator turns on)

1. Apply the resistances to be measured.
2. If using a Series 3700A switch card, perform the following steps to assign a range of channels and assign the channel a DMM configuration:
   a. Using the navigation wheel:
   - Press once to select 3700 card slot number and adjust 1-6.
   - Press a second time to select the start channel number.
   - Press a third time to select the end channel.
   - Press a fourth time to allow DMM configuration assignment to the channel or range of channels.

   b. Press the **CLOSE** key.

1. Press the **TRIG** key and observe the display. If the "Overflow" message is displayed, select a higher range until a normal reading is displayed (or press the **AUTO** key for autoranging). For manual ranging, use the lowest possible range for the best resolution.
2. To measure other switching channels, repeat steps 5 and 6.
3. When finished, press the **OPENALL** key to open all channels.

Resistance measurements through remote interface

Examples of remote interface measurements setups through the remote interface are shown here.

`dmm.func = "twowireohms"` `dmm.autodelay = dmm.ON` `dmm.measurecount = 10` `ReadingBufferOne = dmm.makebuffer(1000)` `dmm.measure(ReadingBufferOne)`	An automatic delay is applied to each measurement when the DMM is measuring two-wire ohms. Take 10 measurements and store them in a reading buffer named ReadingBufferOne that can store up to 1000 readings.

`dmm.func = "fourwireohms"` `dmm.autodelay = dmm.AUTODELAY_ONCE` `dmm.measurecount = 10` `ReadingBufferTwo = dmm.makebuffer(1000)` `dmm.measure(ReadingBufferTwo)`	Sets an auto delay for the first of the ten four-wire ohm readings. Readings two through ten will occur as quickly as possible, with readings stored in a reading buffer called ReadingBufferTwo that can store up to 1000 readings.

Temperature measurements

The Model 3706A can measure temperature using various thermoelectric transducers, including: thermocouples, thermistors, and 3 or 4-wire resistance temperature detectors (RTDs).

When deciding which type to use, note that the thermocouple is the most versatile and useful for significant distances between the sensor and the instrument, the thermistor is the most sensitive, the 4-wire RTD is the most stable, and the 3-wire RTD minimizes the number of conductors per sensor (3).

Settings available for temperature measurements

- aperture (range of 10e-6 s to 0.250 s for 50 Hz; 8.33e-6 s to 0.250 s for 60 Hz)
- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- autozero (`dmm.AUTOZERO_ONCE`, `dmm.ON` or `dmm.OFF`)
- display digits (3, 4, 5, 6, or 7)
- filter count (1 to 100)
- filter enable (`dmm.ON` or `dmm.OFF`)
- filter type (`dmm.FILTER_MOVING_AVG` or `dmm.FILTER_REPEAT_AVG`)
- filter window (0 to 10%)
- Four-wire RTD (`dmm.RTD_PT100`, `dmm.RTD_D100`, `dmm.RTD_F100`, `dmm.RTD_PT385`, `dmm.RTD_PT3916`, `dmm.RTD_USER`) (only with `dmm.transducer` set to `dmm.TEMP_FOURRTD`)
- DMM limit auto clear (`dmm.ON` or `dmm.OFF`)
- DMM limit enable (`dmm.ON` or `dmm.OFF`)
- DMM limit high fail (0 or 1)
- DMM limit high value (–4294967295 to +4294967295)
- DMM limit low fail (0 or 1)

- DMM limit low value (–4294967295 to +4294967295)
- line synchronization (dmm.ON or dmm.OFF)
- math enable (dmm.ON or dmm.OFF)
- math format (dmm.MATH_NONE, dmm.MATH_MXB, dmm.MATH_PERCENT, or dmm.MATH_RECIPROCAL)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- nplc (0.0005 to 15 for 60 Hz; 0.0005 to 12 for 50 Hz)
- offset compensation (dmm.ON or dmm.OFF)
- open detector (dmm.ON or dmm.OFF)
- reference junction (dmm.REF_JUNCTION_SIMULATED, dmm.REF_JUNCTION_INTERNAL, or dmm.REF_JUNCTION_EXTERNAL) (only available when transducer type is set to thermocouple).
- relative offset enable (dmm.ON or dmm.OFF)
- relative offset level
- RTD alpha (0 to 0.01) (only with dmm.transducer set to dmm.TEMP_FOURRTD or dmm.TEMP_THREERTD)
- RTD beta (0 to 1.0) (only with dmm.transducer set to dmm.TEMP_FOURRTD or dmm.TEMP_THREERTD)
- RTD delta (0 to 5) (only with dmm.transducer set to dmm.TEMP_FOURRTD or dmm.TEMP_THREERTD)
- RTD zero (0 to 10000) (only with dmm.transducer set to dmm.TEMP_FOURRTD or dmm.TEMP_THREERTD)
- simulated reference temperature (Celsius (0 °C to 65 °C), Fahrenheit (32 °F to 149 °F), or Kelvin (273 °K to 338 °K)) (only with dmm.transducer set to dmm.TEMP_THERMOCOUPLE)
- thermistor type (2252, 5000, or 10000) (only with dmm.transducer set to dmm.TEMP_THERMISTOR)
- thermocouple type (dmm.THERMOCOUPLE_J, dmm.THERMOCOUPLE_K, dmm.THERMOCOUPLE_T, dmm.THERMOCOUPLE_E, dmm.THERMOCOUPLE_R, dmm.THERMOCOUPLE_S, dmm.THERMOCOUPLE_B, dmm.THERMOCOUPLE_N) (only with dmm.transducer set to dmm.TEMP_THERMOCOUPLE)
- three-wire RTD type (dmm.RTD_PT100, dmm.RTD_D100, dmm.RTD_F100, dmm.RTD_PT385, dmm.RTD_PT3916, dmm.RTD_USER) (only with dmm.transducer set to dmm.TEMP_THREERTD)
- transducer type (dmm.TEMP_THERMOCOUPLE, dmm.TEMP_THERMISTOR, dmm.TEMP_THREERTD, or dmm.TEMP_FOURRTD)
- units (dmm.UNITS_CELSIUS, dmm.UNITS_KELVIN, or dmm.UNITS_FAHRENHEIT)

Autodelay and auto range settings

The following table provides times for autodelay and auto range time for the Model 3706A DMM functions.

For the standard RTD values (PT100, D100, F100, PT385, and PT3916), use 1 kΩ. For user-set RTDs, use 1 kΩ or 10 kΩ, depending on the alpha, beta, delta, and Ro values.

For thermocouples, the autodelay and auto range functions are 1 ms.

For thermistors, for:

- 2252Ω and 5kΩ: 100 MΩ to 10 MΩ, dependent on temperature
- 10kΩ: 1 kΩ to 10 MΩ, dependent on temperature

Thermocouples

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

Type	Range	Resolution
J	-200 °C to +760 °C	0.001 °C
K	-200 °C to +1372 °C	0.001 °C
N	-200 °C to +1300 °C	0.001 °C
T	-200 °C to +400 °C	0.001 °C
E	-150 °C to +1000 °C	0.001 °C
R	0 °C to +1768 °C	0.1 °C
S	0 °C to +1786 °C	0.1 °C
B	+350 °C to +1820 °C	0.1 °C

When two wires made up of dissimilar metals are joined together, a voltage is generated. The generated voltage is a function of temperature. As temperature changes, the voltage changes. The thermocouple voltage equates to a temperature reading. This is the basic operation principle of the thermocouple.

When you connect a thermocouple directly to the input of the Model 3706A, at least one of those connections will be a junction made up of two dissimilar metals. Hence, another voltage is introduced and is algebraically added to the thermocouple voltage. The result will be an erroneous temperature measurement.

To cancel the affects of the unwanted thermal voltage, the thermocouple circuit requires a reference junction that is at a known temperature.

The Model 3706A has an open thermocouple detection circuit. 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 µA pulse of current to the thermocouple before the start of each temperature measurement. For more detail, see Open thermocouple detection (on page 5-19).

NOTE

The default setting is for open thermocouple detection to be on (`dmm.opendetector = dmm.ON`).

Thermocouple connections

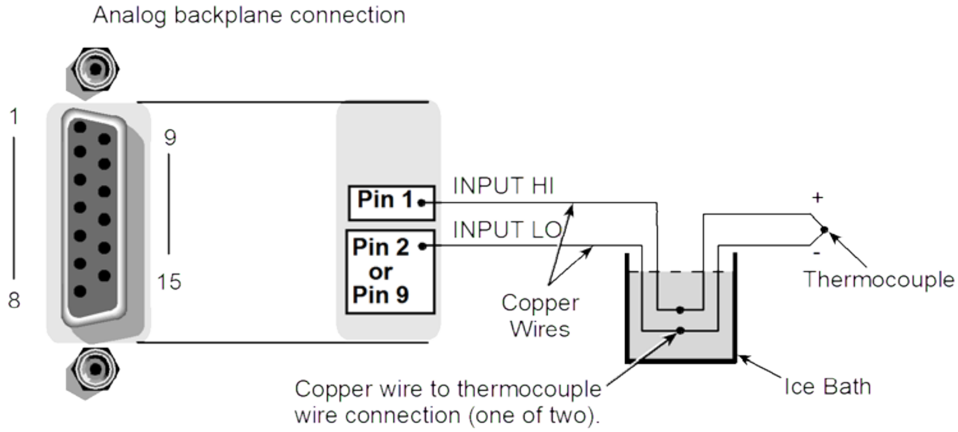
Connections for thermocouples are shown below. Thermocouples are color coded to identify the positive (+) and negative (–) leads (see the table). Note that the negative (–) lead for U.S. type T/Cs is red.

T/C type	Positive (+)	Negative (–)
J	White	Red
	Yellow	Blue
	Red	Blue
	Red	White
	Yellow	Black
K	Yellow	Red
	Brown	Blue
	Red	Green
	Red	White
	Yellow	Purple
N	Orange	Red
	—	—
	—	—
	—	—
	—	—
T	Blue	Red
	White	Blue
	Red	Brown
	Red	White
	Yellow	Blue

T/C type		Positive (+)	Negative (–)
E	U.S.	Purple	Red
	British	Brown	Blue
	DIN	Red	Black
	Japanese	Red	White
	French	Yellow	Blue
R	U.S.	Black	Red
	British	White	Blue
	DIN	Red	White
	Japanese	Red	White
	French	Yellow	Green
S	U.S.	Black	Red
	British	White	Blue
	DIN	Red	White
	Japanese	Red	White
	French	Yellow	Green
B	U.S.	Gray	Red
	British	—	—
	DIN	Red	Gray
	Japanese	Red	Gray
	French	—	—

When using the Model 3706A analog backplane connector, use a simulated reference junction for thermocouple temperature measurements. An ice bath, as shown below, serves as an excellent cold junction because it is relatively easy to hold the temperature to 0 °C. Notice that copper wires are used to connect the thermocouple to the Model 3706A input.

Figure 69: Simulated reference junction



NOTE

The positive lead of the type T thermocouple is made of copper. Therefore, that lead can be connected directly to the input of the switching module (it does not have to be maintained at the simulated reference temperature, in other words, immersed in an ice bath).

For the thermocouple-capable switching modules, you can also use a simulated reference junction as shown, or you can connect the thermocouple wires directly to the screw terminals (internal reference junction). Using a simulated reference junction may be inconvenient, but it will provide more accurate temperature measurements (assuming the user enters a precise reference temperature).

Figure 70: Simulated reference junction switching module

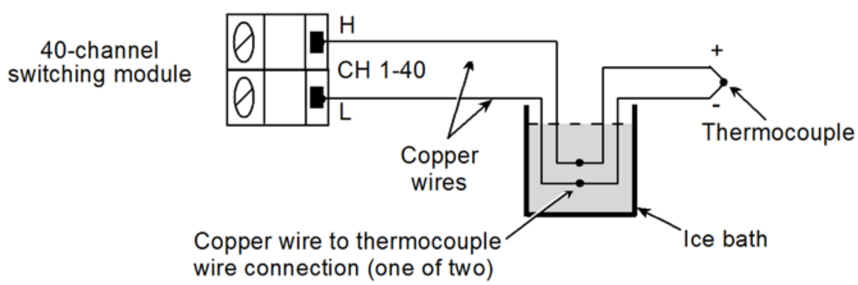
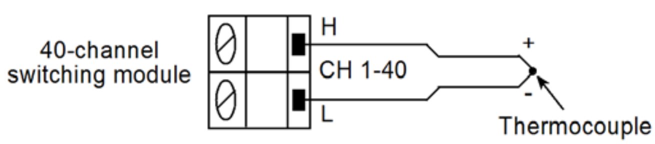


Figure 71: Internal reference junction (40 channel switching module)



Thermocouple measurement from the front panel

NOTE

If the Model 3706A is being controlled remotely, place the instrument in local control by pressing the **EXIT** key.

To make a thermocouple measurement from the front panel:

1. Press the **OPENALL** key to open all switching channels.
2. Select the temperature measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **FUNCTION?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys of the knob to select TEMP.
3. Press the **CONFIG** key and then the **DMM** key. The "TEMP ATTR MENU" opens.
4. Set units of measurement degrees:
   - Turn the navigation wheel to scroll to the "UNITS" menu item (right most menu item) and press the **ENTER** key.
   - Turn the navigation wheel to select desired units (Celsius, Kelvin, or Fahrenheit) and press the **ENTER** key.

1. Set THERMO device attributes:
   - Turn the navigation wheel to scroll to the "THERMO" menu item and press the **ENTER** key.
   - Turn the navigation wheel to scroll to the "THERMOCOUPLE" type and press the **ENTER** key.
   - Turn the navigation wheel to select desired type and press the **ENTER** key.

1. Press the **EXIT** key twice to leave the "TEMP ATTR MENU."
2. Connect the temperature transducers to be measured.
3. If using a switching module, perform the following steps to close the desired channel.
   a. Use the navigation wheel to dial in the channel number.
   b. Press the **CLOSE** key.

1. Press the **TRIG** key and observe the displayed reading.
2. To measure other channels, repeat steps 7 and 8.
3. When finished, press the **OPENALL** key to open all channels.

Thermocouple measurement through the remote interface

To take thermocouple measurements through the remote interface:

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMOCOUPLE` `dmm.thermocouple = dmm.THERMOCOUPLE_J`	Sets the thermocouple type to J.

Thermistors

The temperature measurement range for thermistors is –80 °C to 150 °C (0.01 ° resolution). Thermistor types that are supported include the 2.2 kΩ, 5 kΩ, and 10 kΩ types.

The thermistor is a temperature sensitive resistor. Its resistance changes non-linearly with changes in temperature. Most thermistors have a negative temperature coefficient — as temperature increases, the resistance decreases. The Model 3706A measures the resistance of the thermistor and calculates the temperature reading.

Of all the temperature transducers, the thermistor is the most sensitive. It can detect minute changes in temperature. It is a good choice when measuring slight changes in temperature. The downside for this increased sensitivity is the loss of linearity. Because they are especially non-linear at high temperatures, it is best to use them for measurements below 100 °C.

NOTE

Curve fitting constants are used in the equation to calculate thermistor temperature. The thermistor manufacturer's specified curve fitting may not be exactly the same as the ones used by the Model 3706A.

Thermistor connections

A thermistor can be connected directly to the analog backplane connector or to any of the applicable input channels of a thermistor capable switching module.

Figure 72: Thermistor analog backplane connection

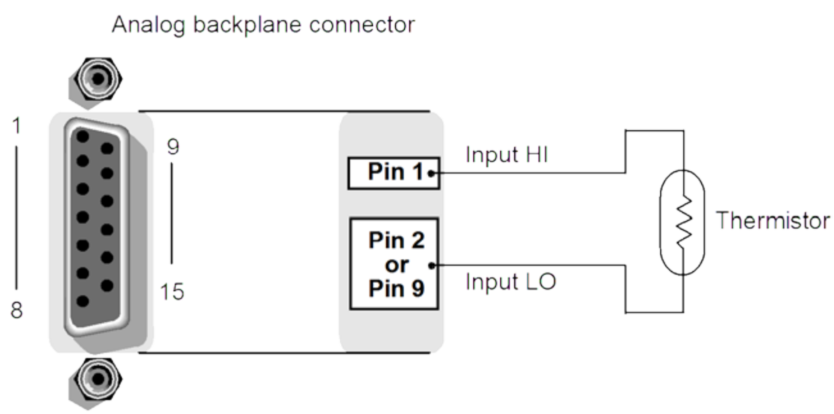
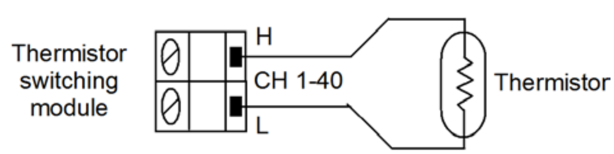


Figure 73: Thermistor switching module connection



Thermistor measurement from the front panel

To configure thermistor measurements from the front panel:

1. If needed, change to the temperature function ("TMP" is displayed) by pressing the **FUNC** key.
2. Select the temperature measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **FUNCTION?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys of the knob to select TEMP.
3. Set units of measurement degrees:
   - Turn the navigation wheel to scroll to the "UNITS" menu item (right most menu item) and press the **ENTER** key.
   - Turn the navigation wheel to select desired units (Celsius, Kelvin, or Fahrenheit) and press the **ENTER** key.

1. Set THERMO device attributes:
   - Turn the navigation wheel to scroll to the "THERMO" menu item and press the **ENTER** key.
   - Turn the navigation wheel to scroll to the "THERMISTOR" temperature connection and press the **ENTER** key.
   - Turn the navigation wheel to select desired resistance (2252Ω, 5000Ω, or 10000Ω) and press the **ENTER** key.
1. Press the **EXIT** key twice to leave the "TEMP ATTR MENU."

Thermistor measurement through the remote interface

This sample remote code configures a thermistor type 2252 and assigns it to a 4-channel scan list.

```
reset()
dmm.func=dmm.TEMPERATURE
dmm.transducer= dmm.TEMP_THERMISTOR          -- or 2
dmm.thermistor=2.252e3
dmm.units=dmm.UNITS_FAHRENHEIT               -- or 4
dmm.configure.set("my_thermist")
dmm.setconfig("4011:4014", "my_thermist")
scan.measurecount=1
buf=dmm.makebuffer(20)
buf.clear()
buf.appendmode=1
scan.create("4011:4014")
scan.scancount=5
scan.execute(buf)
for x=1, buf.n do printbuffer (x,x,buf) end
channel.open("allslots")
```

Also see remote command `dmm.thermistor` (on page 8-240) for more information on setting thermistor measurement attributes.

RTDs (Resistance Temperature Detector)

Of all the temperature transducers, the resistance temperature detector (RTD) exhibits the most stability and linearity. The Model 3706A supports 3-wire and 4-wire RTD types of:

- PT100
- D100
- F100
- PT385
- PT3916

A USER type is also available to modify RTD parameters, such as the resistance at 0°C. The USER type can be enabled from the front panel, but the settings can only be changed using remote programming.

For 4-wire RTDs, the temperature measurement range is –200 °C to 630 °C (0.01 °C resolution).

The RTD has a metal construction (typically platinum). The resistance of the RTD changes with change with temperature. The Model 3706A measures the resistance and calculates the temperature reading. When using default RTD parameters, the resistance of the RTD will be 100 Ω at 0°C.

By default, the Model 3706A performs the 4-wire measurement using offset-compensated ohms, which provides the most accurate way to measure the low resistance of the RTD. For faster RTD measurements when the most accurate measurements are not required, offset-compensation may be disabled for 4-wire RTD measurements.

Use of a 3-wire RTD requires a special math capability to compensate for lead resistance on the 3rd wire.

3-wire RTD connections

Shown below are 3-wire RTD connections to the Model 3706A. For a 3-wire RTD capable 40-channel switching module, paired channels perform the 3-wire measurement. For example, the two input leads of the RTD are connected to a primary channel (1 through 20), while only the LO sense lead is connected to its paired channel (21 through 40). Channel 1 is paired to channel 21, channel 2 is paired to channel 22, and so on.

NOTE

The HI sense of the paired channels are not used for 3-wire RTD.

Figure 74: Three-wire RTD connections

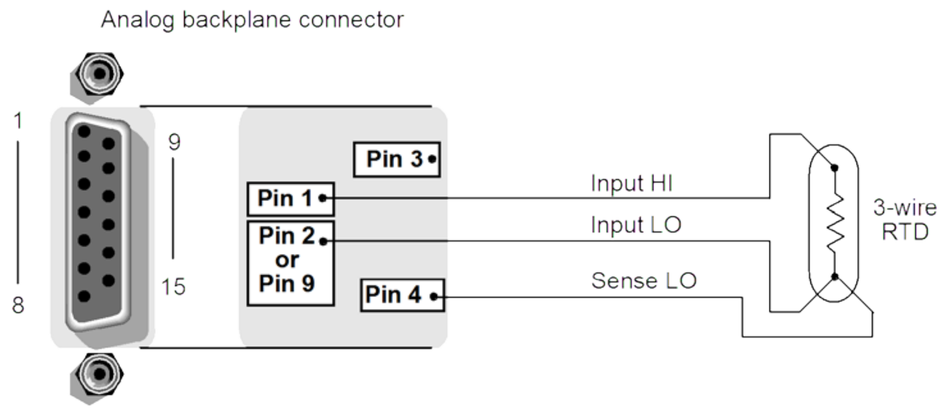
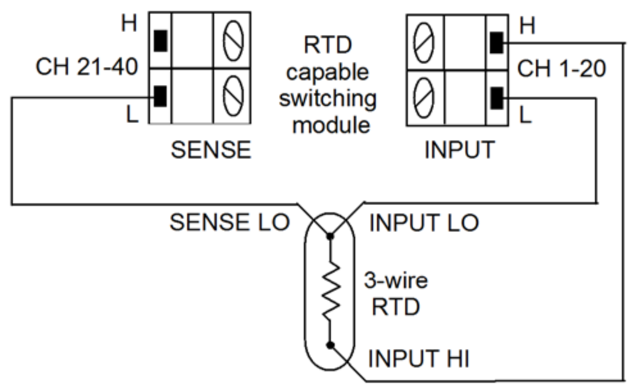


Figure 75: Three-wire RTD switching module connections



4-wire RTD connections

Shown below are 4-wire RTD connections to the Model 3706A. For a 4-wire RTD capable 40-channel switching module, paired channels are used to perform the 4-wire measurement. For example, the two input leads of the RTD are connected to a primary channel (1 through 20), while the two sense leads are connected to its paired channel (21 through 40). Channel 1 is paired to channel 21, channel 2 is paired to channel 22, and so on.

Figure 76: Four-wire RTD connections

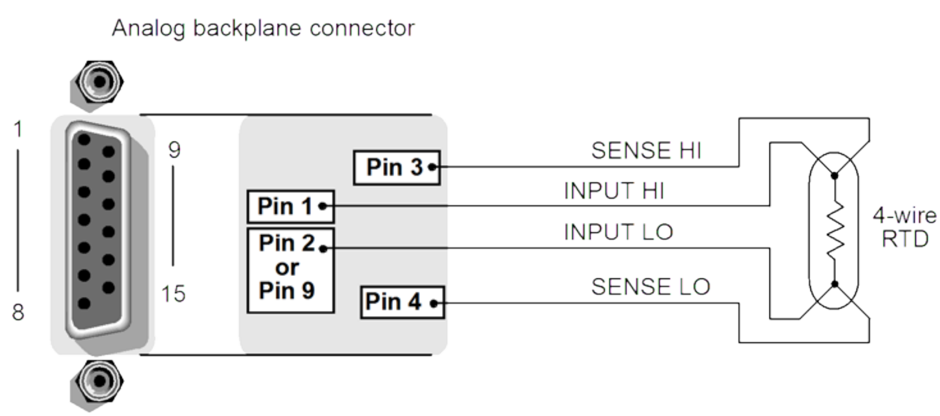
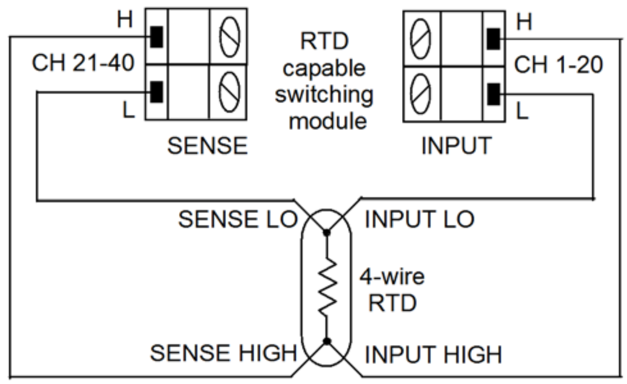


Figure 77: Four-wire RTD switching module connections



RTD temperature measurement configuration

The alpha, beta, delta, and Ω at 0 °C parameters for the five basic RTD types are provided in the table below.

NOTE

These parameters can be modified using remote commands for USER type RTDs.

Type	Standard	Alpha	Beta	Delta	Ω at 0 °C
PT100	ITS-90	0.00385055	0.10863	1.49990	100
D100	ITS-90	0.003920	0.10630	1.49710	100
F100	ITS-90	0.003900	0.11000	1.49589	100
PT385	IPTS-68	0.003850	0.11100	1.50700	100
PT3916	IPTS-68	0.003916	0.11600	1.50594	100

RTD measurement from the front panel

To configure 3 or 4-wire RTD measurements from the front panel:

1. If needed, change to the temperature function ("TMP" is displayed) by pressing the **FUNC** key.
2. Select the temperature measurement function by pressing the **CONFIG** key, and then pressing the **DMM** key. FUNC flashes on, then off. Press the **ENTER** key or wheel. **FUNCTION?** is displayed on the first line of the display and the second line displays available functions. Use the left or right arrow keys of the knob to select TEMP.
3. Set units of measurement degrees:
   - Turn the navigation wheel to scroll to the "UNITS" menu item (right most menu item).
   - Press the **ENTER** key.
   - Using the navigation wheel, select desired units (Celsius, Kelvin, or Fahrenheit).
   - Press the **ENTER** key.

1. Set three-wire or four-wire RTD device attributes:
   - Turn the navigation wheel to scroll to the "THERMO" menu item and press the **ENTER** key.
   - Turn the navigation wheel to scroll to the "THREERTD" or "FOURRTD" temperature connection and press the **ENTER** key.
   - Turn the navigation wheel to select desired RTD type (PT100, D100, F100, PT3916, PT385, or USER) and press the **ENTER** key.
1. Press the **EXIT** key twice to leave the "TEMP ATTR MENU."
2. Connect the temperature transducers to be measured.
3. If using a switching module, perform the following steps to close the desired channel. Note that for 3 or 4-wire RTD measurements, you will close the primary (INPUT) channel (1 through 10). The channel that it is paired to will close automatically.
   a. Use the navigation wheel to dial in the channel number.
   b. Press the **CLOSE** key.
1. Press the **TRIG** key and observe the displayed reading.
2. To measure other switching channels, repeat steps 6 and 7.
3. When finished, press the **OPENALL** key to open all channels.

RTD measurement from the remote interface

You can use the remote command dmm.fourrtd (on page 8-186) or dmm.threertd (on page 8-242) to set attributes.

For example, the following remote commands configure temperature function to a custom RTD and assign it to a 10-channel scan list.

```
reset()
dmm.func=dmm.TEMPERATURE
-- or 3, or dmm.TEMP_FOURRTD, or 4
dmm.transducer= dmm.TEMP_THREERTD
-- dmm.fourrtd also supported
dmm.threertd=dmm.RTD_USER
-- allowed values are 0 to 0.01
dmm.rtdalpha= 0.003
-- allowed values are 0 to 1.00
dmm.rtdbeta= 0.105
-- allowed values are 0 to 5.00
dmm.rtddelta = 1.51
-- allowed values are 0 to 10,000
dmm.rtdzero= 125
-- default dmm.ON
dmm.offsetcompensation=dmm.OFF
dmm.configure.set("my_rtd_user")
dmm.setconfig("4001:4010", "my_rtd_user")
scan.measurecount=1
buf=dmm.makebuffer(20)
buf.clear()
buf.appendmode=1
scan.create("4001:4010")
scan.scancount=2
scan.execute(buf)
for x=1, buf.n do printbuffer (x,x,buf) end
channel.open("allslots")
```

Frequency and period measurements

The Model 3706A is specified for frequency measurements from 3 Hz to 500 kHz on voltage ranges of 100 mV, 1 V, 10 V, 100 V, and 300 V. Period (1/ frequency) measurements can be taken from 2 µs to 333 ms on the same voltage ranges as the frequency.

the input impedance is 1 MΩ || less than 100 pF, AC coupled.

The instrument uses the volts input to measure frequency. The AC voltage range can be changed with the **RANGE** ▼ and ▲ keys. The signal voltage must be greater than 10% of the full-scale range.

NOTE

Frequency or period measurements as low as 0.5 Hz (2 seconds) and less than or equal to 1 MHz (1 µs) are possible but depend on the selected range.

⚠ CAUTION

Do not apply more than maximum input levels indicated or instrument damage may occur.The voltage limit is subject to the 8 x 10^7 VHz product.

Settings available for frequency and period measurements

- aperture (range of 0.01 s to 0.273 s)
- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- autozero (`dmm.AUTOZERO_ONCE`, `dmm.ON` or `dmm.OFF`)
- display digits (3, 4, 5, 6, or 7)
- DMM limit auto clear (`dmm.ON` or `dmm.OFF`)
- DMM limit enable (`dmm.ON` or `dmm.OFF`)
- DMM limit high fail (0 or 1)
- DMM limit high value (–4294967295 to +4294967295)
- DMM limit low fail (0 or 1)
- DMM limit low value (–4294967295 to +4294967295)
- math enable (`dmm.ON` or `dmm.OFF`)
- math format (`dmm.MATH_NONE`, `dmm.MATH_MXB`, `dmm.MATH_PERCENT`, or `dmm.MATH_RECIPROCAL`)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- relative offset enable (`dmm.ON` or `dmm.OFF`)
- relative offset level
- threshold (0 to 303 V)

Autodelay and auto range settings

The following table provides times for autodelay and autorange time for the Model 3706A DMM functions.

Function	Ranges and delays					
Frequency and periods	**Range**	100 mV	1 V	10 V	100 V	300 V
	Autodelay	100 ms				
	Autorange	100 ms				

Trigger level

Frequency and period use a zero-crossing trigger, meaning that a count is taken when the frequency crosses the zero level. The Model 3706A uses a reciprocal counting technique to measure frequency and period. This method generates constant measurement resolution for any input frequency. The multimeter's AC voltage measurement section performs input signal conditioning. If the input signal voltage exceeds the selected voltage range, 000.0000 mHz (0.000000 µs) will be returned.

Gate time

The gate time is the amount of time the Model 3706A uses to sample frequency or period readings.The gate time can be set from 0.01 to 0.273 s by setting the DMM aperture attribute.

The Model 3706A completes a reading when it receives its first positive zero-crossing after the gate time expires. For example, for any arbitrary frequency, you may wait up to the gate time plus two times the period of the input waveform before the Model 3706A returns a reading.

Frequency connections

Frequency connections for the Model 3706A and a switching module are shown below.

Figure 78: FREQ and PERIOD input connections

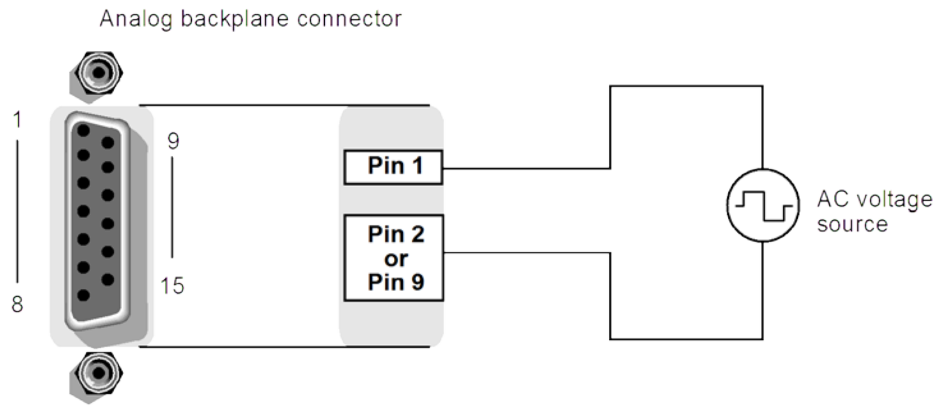
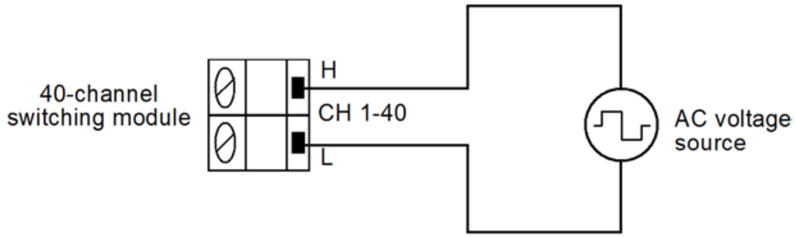


Figure 79: FREQ and PERIOD connections (switching module)



Frequency and period measurement procedure from front panel

CAUTION

Do not apply more than the maximum input levels for the Model 3706A or installed switching module (whichever is lower) or instrument damage may occur.

NOTE

If the Model 3706A is being controlled remotely, place the instrument in local control by pressing the **LOCAL** or **EXIT** key.

1. Press the **OPENALL** key to open all switching channels.
2. Select the **CONFIG** key, and then select the **DMM** key. Select the FUNC menu by pressing **ENTER**. Scroll through the menu until FREQ or PERIOD is displayed, using the navigation wheel or left right arrows.
3. Set threshold voltage:
   - Turn the navigation wheel to scroll to the "THRESHOLD" menu item (right most menu item) and press the **ENTER** key.
   - Using the navigation wheel, dial in the desired voltage to be used as a threshold (0 V to 303 V; default is the 10 V range).
   - Press the **ENTER** key to set.
   - Press the **EXIT** key to leave the "FREQ ATTR MENU."

1. Apply the AC voltages to be measured (see CAUTION).

NOTE

When observing the displayed readings, if 000.0000 mHz or 000.0000 ms is displayed, select a lower range until a normal reading is displayed. Use the lowest possible range for the best resolution.

2. Press the **TRIG** key and observe the displayed reading.
3. To measure other switching channels, repeat steps 5 and 6.
4. When finished, press the **OPENALL** key to open all channels.

Frequency and period measurement procedure through remote interface

CAUTION

Do not apply more than the maximum input levels for the Model 3706A or installed switching module (whichever is lower) or instrument damage may occur.

To set the frequency through the remote command interface:

`dmm.func = "frequency"` `dmm.threshold = 30`	Sets the threshold range for frequency to 30.

Continuity testing

The Model 3706A can test continuity using the 2-wire 1 kΩ range with a user selectable threshold resistance level (1 Ω to 1000 Ω). 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 math calculations (see "Settings available for continuity testing" on page 4-38), with b as an offset, to compensate for cable and 3700A card path resistance.

NOTE

The reading rate for continuity is fixed at 0.006 power line cycles. Limits and digital outputs cannot be used when testing continuity with the continuity (CONT) function. If you need to use these operations, use the two-wire ohm function to test continuity.

Settings available for continuity testing

- autodelay (`dmm.AUTODELAY_ONCE`, `dmm.ON` or `dmm.OFF`)
- display digits (3, 4, 5, 6, or 7)
- line synchronization (`dmm.ON` or `dmm.OFF`)
- math enable (`dmm.ON` or `dmm.OFF`)
- math format (`dmm.MATH_NONE`, `dmm.MATH_MXB`, `dmm.MATH_PERCENT`, or `dmm.MATH_RECIPROCAL`)
- math mxb b factor (–4294967295 to +4294967295)
- math mxb m factor (–4294967295 to +4294967295)
- math mxb units
- math percent (–4294967295 to +4294967295)
- nplc (0.0005 to 15 for 60 Hz; 0.0005 to 12 for 50 Hz)
- threshold (1 to 1000 Ω)

Autodelay and auto range settings

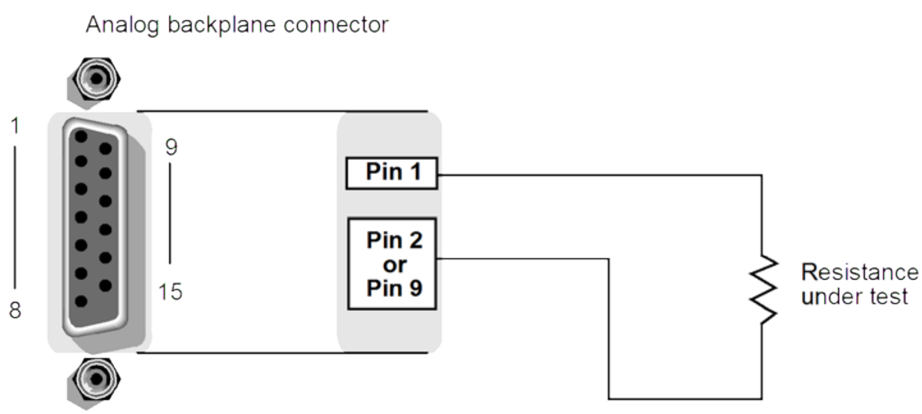
The following table provides times for autodelay and auto range time for the Model 3706A DMM functions.

Function	Detector bandwidth	Range and delays
	Range	1kΩ
Continuity	Autodelay	3ms
	Autorange	2.5ms

Continuity testing connections

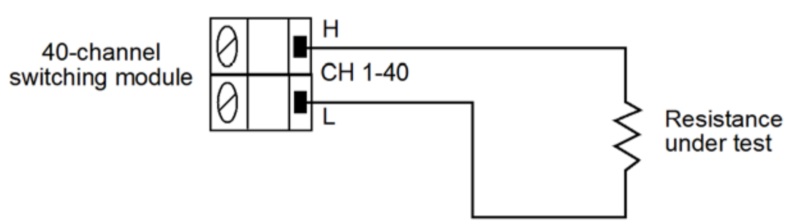
When using the rear analog backplane connector, connect the test leads to the INPUT HI and LO terminals as shown below.

Figure 80: Continuity connections



Connections to test continuity using a switching module are shown below. Because this is a 2-wire ohm measurement, channels 1 through 40 of a 40-channel switching module can be used.

Figure 81: Continuity connections using a switching module



Continuity testing procedure

NOTE

If the Model 3706A is in remote, place the unit in local by pressing the **LOCAL** key (or the **EXIT** key). Refer to DCV measurements for function configuration and menus.

1. Select the continuity measurement function:
   a. Press the **CONFIG** key, and then press the **DMM** key. `FUNC` flashes on, then off.
   b. Press the **ENTER** key or navigation wheel. The "Function?" menu is displayed.
   c. Turn the navigation wheel to highlight **CONT**, then push the navigation wheel to select it.
   d. Press the **EXIT** key to leave the "Function?" menu.
1. Set threshold resistance:
   e. Press the **CONFIG** key, and then press the **DMM** key. `FUNC` flashes on, then off.
   f. Turn the navigation wheel to scroll to the **THRESHOLD** menu item, and then press the **ENTER** key.
   g. Using the <mw>, dial in the desired resistance to be used as a threshold (1 Ω to 1000 Ω).
   h. Press the **ENTER** key to set.
   i. Press the **EXIT** key to leave the "CONT ATTR MENU."
1. Apply the resistance to be tested. If using a switching module, perform the following steps to close the appropriate channel.
   j. Use the navigation wheel to dial in the channel number.
   k. Press the **CLOSE** key.
1. Press the **TRIG** key and observe the displayed reading.
2. To measure other switching channels, repeat steps 5 and 6.

NOTE

If the measured circuit is below the set threshold level, the instrument will display the resistance readings. If the measured circuit is above the threshold level, the instrument will display the message "OPEN."

3. To disable continuity testing, select a different function (for example, DCV).
4. When finished, press the **OPENALL** key to open all channels.

NOTE

Limits and digital outputs cannot be used when testing continuity with the continuity (CNT) function. If you need to use these operations, use the 2WΩ function to test continuity.

Also see the bus command `dmm.threshold` (on page 8-243) for more information on threshold attributes.

`dmm.threshold` is a common command. To enable a unique continuity threshold, first select the function `dmm.func = "continuity"`, then select the threshold value. The threshold value will be remembered after exiting when returning to the function (unless reset).

Refining measurements

Relative offset

You can use the relative offset (REL) feature to set offsets to zero (0) or subtract a baseline reading from present and future readings. With relative offset enabled, subsequent readings are the difference between the actual input value and the relative offset value, as follows:

Displayed reading = Actual input – Relative offset value

Once a relative offset value is established for a measurement function, the value is the same for all ranges.

When relative offset is enabled, the REL indicator turns on. Changing measurement functions changes the relative offset value to the established relative offset value and state for that measurement function.

The various instrument operations, including relative offset, are performed on the input signal in a specific, predetermined order. For example, if both relative offset and a math operation are enabled, the relative offset operation is always performed before the math operation.

Set relative offset from the front panel

You can set a relative offset through front panel by acquiring the offset or by setting an offset manually.

To set and enable a relative offset through the front panel by acquiring the offset:

1. Select the desired measurement function and an appropriate range setting.
2. Apply the signal to which you want to apply a relative offset to a switching channel input or to the Model 3706A inputs.
3. If you are using a switching module, close the input channel. (see Operation keys for basic information about the front panel user interface).
4. Press the **REL** key to acquire the relative offset value. The REL annunciator turns on. The displayed value will not become zero until a new reading is triggered.
5. Apply the signal to be measured. The relative offset value is subtracted from the next reading that is triggered.

To set and enable a relative offset through the front panel manually:

1. Select the desired measurement function and an appropriate range setting.
2. If you are using a switching module, close the input channel. (see Operation keys for basic information about the front panel user interface).
3. Press **CONFIG**, then press the **REL** key. The Relative Offset Menu is displayed.
4. Select **ENABLE**.
5. Select **Yes**. The REL annunciator turns on.
6. Press **EXIT** to return to the Relative Offset Menu.
7. Select **LEVEL**.
8. Set the relative offset value as needed.
9. Apply the signal to be measured. The relative offset value is subtracted from the next reading that is triggered.

NOTE

If you press the **REL** key, the manually entered value will be overwritten with an acquired value.

To disable the relative offset function:

Press **REL** a second time to disable the relative offset function.

NOTE

You can perform the equivalent of relative offset manually by using the mX+b (on page 4-44) math function. Set *m* to 1 and *b* to the value of the offset.

Set relative offset through the remote interface

The `dmm.rel.level()` command is used to specify the relative offset value for the active function.

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

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

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

To manually set a relative offset value of 1.5 µV, use this command sequence:

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

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

Scanning

When a scan is configured, each channel can have its own unique relative offset value. For remote programming, the channel list parameter is used to configure channels for a scan.

For example:

To attach a 1 µV relative offset level to a configuration, send the following commands:

```
-- Select DC volts function.
dmm.func = "dcvolts"
-- Reset DC volts only.
dmm.reset("active")
-- Set the relative offset level.
dmm.rel.level=1e-6
-- Enable relative offset.
dmm.rel.enable = dmm.ON
-- Call the configuration myconfig.
dmm.configure.set("myconfig")
-- Set channels 1001 to 1030 to use myconfig configuration.
dmm.setconfig("1001:1030", "myconfig")
-- Create scan list of channels 1001 to 1030 using myconfig.
scan.create("1001:1030")
```

Math calculations

The Model 3706A has three built-in math calculations:

- mX+b
- percent
- reciprocal (1/X)

NOTE

The various instrument operations, including Math, are performed on the input signal in a specific, predetermined order. For example, if both relative offset and a math operation are enabled, the relative offset operation is always performed before the math operation.

Basic math operation

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

mX+b

The mx+b math operation lets you manipulate normal display readings (x) mathematically according to the following calculation:

$mx + b = Y$

Where:

- **m** is a user-defined constant for the scale factor
- **x** is the measurement reading (if you are using a relative offset, this is the measurement with relative offset applied)
- **b** is the user-defined constant for the offset factor
- **Y** is the displayed result

When the mx+b math operation is active, the unit of measure for the front-panel readings is **X**. 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 will be the difference between the actual input and the offset value.

Set mX+b units from the front panel

The attribute for mX+b must be one character. It can be any letter of the alphabet, the degrees symbol (°), the micro symbol (μ), or the ohms symbol (Ω).

To set mX+B units from the front panel:

NOTE

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

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

Set mX+b units through the remote interface

You can set mX+b units through the remote interface with the command `dmm.math.mxb.units`. The attribute for `dmm.math.mxb.units` must be one character enclosed in single or double quotes. It can be any letter of the alphabet, the degrees symbol (°), the micro symbol (µ), or the ohms symbol (Ω).

The ohm symbol (Ω), the micro symbol (µ), and the degree symbol (°) are not ASCII characters and must be substituted with the ']', '[' and '\' characters. Valid characters are therefore:

- **A** to **Z**
- **]** for ohms
- **[** for microvolts
- **** for degrees

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

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

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

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

To use the degrees symbol (°) as units designator:

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

NOTE

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

Use dmm.math.mxb.bfactor (on page 8-209) and dmm.math.mxb.mfactor (on page 8-210) to set the b and m factor for mX+b.

Once all settings are configured, set dmm.math.enable (on page 8-206) to `dmm.ON` to enable math operation.

NOTE

For more detail, see dmm.math.mxb.units (on page 8-211).

Percent

The percent math function displays measurements as percent deviation from a specified constant. The percent calculation is:

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

Where:

Percent: The result

Input: The measurement (if relative offset is being used, this is the relative offset value)

Reference: The user-specified constant

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

Set percent from the front panel

To set a percent value on the front panel:

1. Open the function attribute menu:
   - Press the **CONFIG** key.
   - Press the **DMM** key.

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

Set percent through the remote interface

The `dmm.math.percent` attribute (see dmm.math.percent (on page 8-212)) specifies the reference value for the percent calculation, while the `dmm.rel.acquire` function (see dmm.rel.acquire (see "dmm.rel.acquire()" on page 8-224)) uses the input signal as the reference value.

The acquire function triggers a single reading and uses the result as the new relative offset value. When a value is set using dmm.math.percent (on page 8-212), the dmm.math.percent (on page 8-212) query command returns the programmed value. When reference is set using dmm.rel.acquire() (on page 8-224), the dmm.math.percent (on page 8-212) query command returns the acquired reference value.

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

```
-- Set percent to 5
dmm.math.percent = 5
-- Sends 5 to the computer for display
print(dmm.math.percent)
```

You can use the relative offset used to set the percent as follows:

```
-- Sets percent with relative offset acquire value.
dmm.math.percent = dmm.rel.acquire()
-- Send the result of relative offset acquire to a computer.
print(dmm.math.percent)
```

Reciprocal (1/X)

You can set math operation to reciprocal to display the reciprocal of a reading.

The reciprocal is 1/X, where X is the reading. If relative offset is on, the 1/X calculation uses the input signal with the relative offset applied.

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

The result of the 1/X calculation may be displayed in exponential notation. For example, a displayed reading of +2.500E+03 R is equivalent to 2500 R (2.5K R).

Example:

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

```
0.400000 R
```

Scanning

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

Example:

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

```
-- Select DC volts function.
dmm.func = "dcvolts"
-- Reset DC volts only.
dmm.reset("active")
-- Set the math operation to be reciprocal for measurements.
dmm.math.format = dmm.MATH_RECIPROCAL
-- Enable the math operation for measurements.
dmm.math.enable = dmm.ON
-- Call the configuration mymath.
dmm.configure.set("mymath")
-- Set Channels 1001 to 1030 to use the mymath configuration.
dmm.setconfig("1001:1030", "mymath")
-- Create scan list of channels 1001 to 1030 using mymath.
scan.create("1001:1030")
```

dB commands

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

$$dB = 20\log \frac{Vin}{Vref}$$

Where:

V$_{IN}$: DC or AC input signal.

V$_{REF}$: Specified voltage reference level.

The instrument will read 0 dB when the reference voltage level is applied to the input. If a relative value is in effect when dB is selected, the value is converted to dB, and then 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 dB calculation takes the absolute value of the ratio V_{IN}/V_{REF}. The largest negative value of dB is -160 dB. This will accommodate a ratio of V_{IN} = 1 µV and V_{REF} = 1000 V.

dB configuration

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

To select UNITS from the front panel, while the active DMM function is DCV or ACV:

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

To select dB configuration over the remote interface:

Set the active DMM function to DCV or ACV and set `dmm.dbreference`. For example:

```
dmm.func = "dcvolts"
dmm.dbreference = 5
```

dB scanning

Each channel in a scan may be configured to use dB.

Create a configuration that has dB enabled for units for the desired function by using the dmm.configure.set (see "dmm.configure.set()" on page 8-175) command. Once the configuration exists, use the dmm.setconfig() (on page 8-237) command to connect the configuration to the desired channels. Now the channels can be added to scanning (see scan.create() (on page 8-325) and scan.add() (on page 8-319) commands). To remotely control the units for AC and DC volts, use the dmm.units (on page 8-245) command.

Range

You can use the range to set an expected measurement value. The instrument will select the range appropriate to measure that value.

If you set a range, the autorange feature is automatically disabled. The instrument selects the range to best match the expected measure value for the functions, as shown below.

The range setting is saved with the DMM function setting, so if you select another function, then return to the previous function, the range settings you set previously are retained.

You cannot select a range that includes different channel types.

NOTE

A power cycle or an instrument reset will clear the saved ranges.

Measurement ranges and maximum readings

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

Function	Ranges	Maximum reading
DCV (DC voltage)	100 mV, 1 V, 10 V, 100 V, 300 V	± 303 V
ACV (AC voltage)	100 mV, 1 V, 10 V, 100 V, 300 V	± 303 V
DCI (DC current)	10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 3 A	± 3.1 A
ACI (AC current)	1 mA, 10 mA, 100 mA, 1 A, 3 A	± 3.1 A
Ω2 (2-wire ohm)	10 Ω, 100 Ω, 1 kΩ, 10 kΩ, 100 kΩ, 1 MΩ, 10 MΩ, 100 MΩ	120 MΩ
Ω4 (4-wire ohm)	1 Ω, 10 Ω, 100 Ω, 1 kΩ, 10 kΩ, 100 kΩ, 1 MΩ, 10 MΩ, 100 MΩ	120 MΩ
Ω4 OC (4-wire ohm offset compensated)	1 Ω, 10 Ω, 100 Ω, 1 kΩ, 10 kΩ	12 kΩ
Ω4 DRY+ (4-wire ohm dry circuit)	1 Ω, 10 Ω, 100 Ω, 1 kΩ, 10 kΩ	2.4 kΩ
TMP (temperature)	–200 °C to 1820 °C	Sensor dependent
FREQ (frequency)	100 mV, 1 V, 10 V, 100 V, 300 V	3 Hz to 500 kHz
PER (period)	100 mV, 1 V, 10 V, 100 V, 300 V	2 µs to 333 ms
CNT (continuity)	1 kΩ Threshold adjustable 1 Ω to [illegible]	
CSΩ (commonside ohm)	1 Ω, 10 Ω, 100 Ω, 1 kΩ, 10 kΩ, 100 kΩ, 1 MΩ, 10 MΩ, 100 MΩ	120 MΩ

Temperature

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

Select a range from the front panel

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

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

Select range through the remote interface

To select a range through the remote interface, specify the expected reading as an absolute value for the `dmm.range` (on page 8-222) command. The Model 3706A will then go to the most sensitive range for that expected reading.

For example, if you expect a reading of approximately 3 V, send:

```
dmm.range = 3
```

The instrument will select the 10 V range.

Set up autoranging from the front panel

To enable autorange, press the **AUTO** key. The AUTO indicator turns on when autoranging is selected. While autoranging is enabled, the instrument automatically selects the best range at which to measure the applied signal.

Autoranging should not be used when optimum speed is required.

NOTE

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

Up-ranging occurs at 120% of range. The Model 3706A will down-range when the reading is less than 10% of nominal range.

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

Autoranging is automatically disabled when you select a specific range by pressing the ▲ or ▼ key or by sending the remote command `dmm.range`.

Set up autorange through the remote interface

Autorange is enabled by setting the `dmm.autorange` attribute (see dmm.autorange (on page 8-155)) to either `dmm.ON` or `1`.

When autorange is enabled, the range is changed automatically for the selected range value.

To disable autorange, either set the `dmm.autorange` attribute to `dmm.OFF` or `0`, or send a valid `dmm.range` attribute (see dmm.range (on page 8-222)).

When autorange is disabled, the instrument remains at the selected range.

Scanning

Each channel of scan configuration can be associated with a unique digital multimeter (DMM) configuration (which includes a range setting). When a scan completes, the DMM remains in the configuration associated with the last completed measurement step. For remote programming, the channel list parameter is used to configure channels for a scan.

See dmm.configure.set() (on page 8-175), dmm.setconfig() (on page 8-237), scan.create() (on page 8-325), and scan.add() (on page 8-319) commands, and the Scanning and triggering (on page 3-1) section for more details, including front panel operation.

Optimizing measurement speed

Rate

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

The integration time is specified in parameters based on a number of power line cycles (NPLC), where 1 PLC for 60 Hz is 16.67 ms (1/60) and 1 PLC for 50 Hz is 20 ms (1/50).

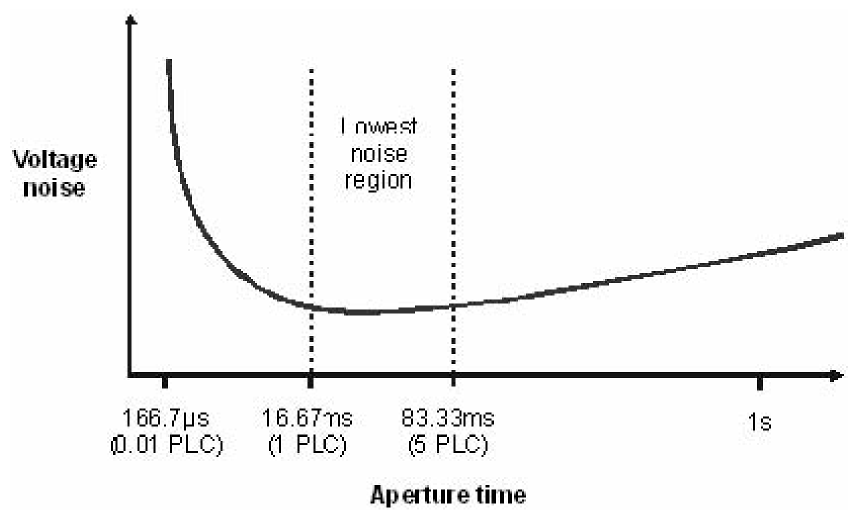
In general, the fastest integration time is 0.1 PLC using the front panel **RATE** key, or 0.0005 PLC from the remote interface or through the **DMM > CONFIG NPLC** menu. This results in increased reading noise and fewer usable digits.

The slowest integration time is 5 PLC using the front panel **RATE** key, or 15 PLC from the remote interface or through the **DMM > CONFIG NPLC** menu. This provides the best common-mode and normal-mode rejection.

In-between settings are a compromise between speed and noise.

The Model 3706A has a parabola-like shape for its speed versus noise characteristics. The Model 3706A is optimized for the 1 PLC to 5 PLC reading rate. At these rates (lowest noise region in graph), the Model 3706A 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 82: Speed compared to noise characteristics



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

Rate cannot be set for continuity. The continuity rate is fixed at 0.006 PLC.

NOTE

The Model 3706A 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.

Setting Rate from the front panel

The **RATE** key sets measurement speed from the front panel. Press the **RATE** key until the speed message is displayed. The second line of the display contains the NPLC setting.

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

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

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

A summary of the rate settings are shown in the following table.

Function	**Slow**	**Medium**	**Fast**
DCV, DCI	NPLC=5	NPLC=1	NPLC=0.1
ACV, ACI	NPLC=X, BW=3	NPLC=X, BW=30	NPLC=1, BW=300
Ω2, Ω4, CSΩ	NPLC=5	NPLC=1	NPLC=0.1
FREQ, PERIOD	APER=0.250s	APER=0.1s	APER=0.01s
Continuity	X	X	NPLC=0.006
NOTES: NPLC = Number of power line cycles. BW = Bandwidth (in Hz). APER = Aperture in seconds. X = Setting ignored (fixed NPLC).			

You can also set the rate using the NPLC option in the function attribute menu. Press **CONFIG**, then **DMM** to display the function attribute menu. From the function attribute menu, select **NPLC** to select a specific value for NPLC.

Setting measurement speed from a remote interface

Use the dmm.aperture (on page 8-150) or dmm.nplc (on page 8-217) command to set the measurement speed (integration time) through the remote interface.

NOTE

For `dmm.nplc` settings that are less than 0.2 power line cycles, sending `dmm.AUTOZERO_ONCE` results in significant delays. For example, the delay time at a NPLC of 0.0005 is 2.75 s. The delay time at 0.199 is 5.45 s.

Bandwidth

There are three bandwidth settings for AC volt and AC current measurements. The RATE setting determines the bandwidth setting as follows:

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

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

When the medium bandwidth is chosen, 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.

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

To achieve the best accuracy for AC volt 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.

NOTE

A rate command (dmm.nplc (on page 8-217) or dmm.aperture (on page 8-150)) for AC volts and AC current is only valid if the bandwidth for that AC function is set to 300 (300 Hz to 300 kHz). Bandwidth is set using the dmm.detectorbandwidth (on page 8-179) remote command or the DETECTBW menu option under the function's attribute menu.

DC voltage, DC current, and resistance measurement speed

To optimize measurement speed, select:

- `dmm.autozero=dmm.OFF`
- `dmm.autodelay=dmm.OFF`
- `dmm.nplc=0.0005`
- `dmm.filter=dmm.OFF`
- `dmm.autorange=dmm.OFF`
- `dmm.measurecount>=1000`

For resistance, assumed two-wire ohm.

AC voltage and AC current optimize speed

Select:

- `dmm.detectorbandwidth=300`
- `dmm.autodelays=dmm.OFF`
- `dmm.autozero=dmm.OFF`
- `dmm.autorange=dmm.OFF`
- `dmm.filter=dmm.OFF`
- `dmm.nplc=0.0005`

Temperature optimize measurement speed

Select:

- `dmm.transducer=dmm.TEMP_THERMOCOUPLE`
- `dmm.opendetector=dmm.OFF`
- `dmm.nplc=0.0005`
- `dmm.autozero=dmm.OFF`
- `dmm.filter=dmm.OFF`
- `dmm.autodelay=dmm.OFF`

Optimizing measurement accuracy

The following two charts represent root-mean-square (RMS) noise versus aperture time (or NPLC) and reading rate versus aperture time (or NPLC). Refer to these charts when selecting best accuracy at a given reading rate. Generally, increasing the aperture time reduces the RMS noise. For aperture times more than 100 ms or 5 power line cycles, thermal offsets can increase the RMS noise.

Figure 83: Readings Rate versus Aperture Time

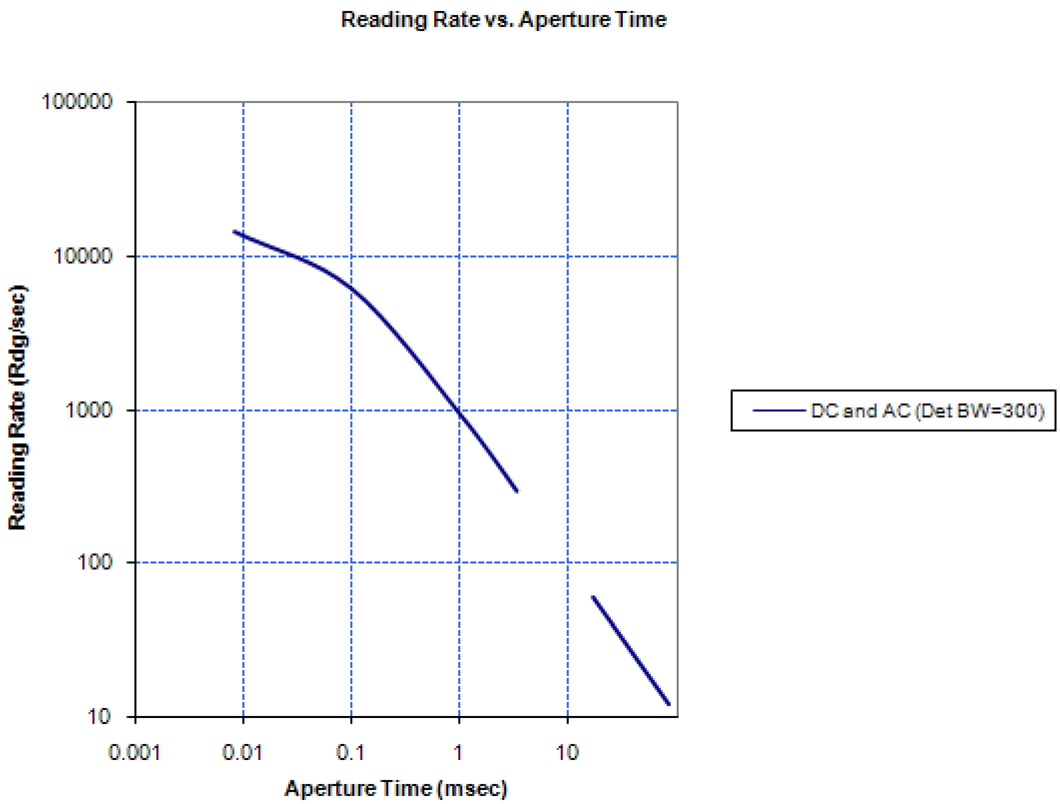
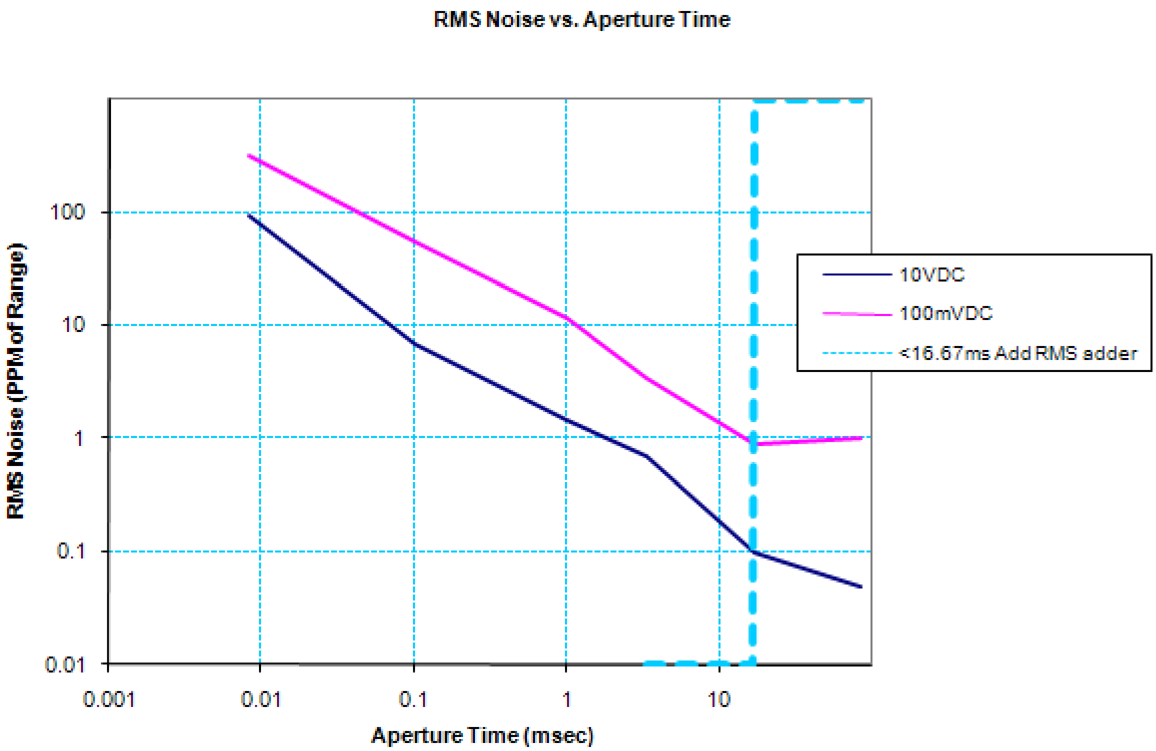


Figure 84: RMS Noise vs. Aperture Time



DC voltage, DC current, and resistance measurement accuracy

To optimize measurement accuracy:

- 1 or 5 NPLC, filter off, fixed range.
- Use relative offset on DC voltage and 2-wire resistance measurements when appropriate.
- Use four-wire, offset compensation on, and line sync on for resistance measurements, especially through a 3700A switch card, for best accuracy.

AC voltage and AC current optimize accuracy

Select Detectorbandwidth 3, autodelays On, and fixed range.

Temperature optimize accuracy

1 or 5 NPLC.

Voltage

DC volts input divider

Normally, the input resistance for the 100 mVDC, 1 VDC, and 10 VDC ranges is more than 10 GΩ. You can set the input resistance for the three lower DC volt ranges to 10 MΩ by enabling the input divider.

When you enable the input divider:

- The measurement INPUT HI is connected to INPUT LO
- Some external devices (such as a high voltage probe) must be terminated to a 10 MΩ load
- The measurement of open leads is maintained near 0 V
- Internal I_{BIAS} through the 10 MΩ causes an open input to read less than – 0.4 mV. With a short circuit (and the input divider on or off), the short circuit to read less than ±0.9 µV.

The input divider can be enabled from the front panel when function is "dcvolts" by pressing the **CONFIG** key, then the **DMM** key.

To control the divider over the remote interface, use the dmm.inputdivider (on page 8-190) command.

Low level considerations

For sensitive measurements, external considerations affect the accuracy. Effects that are not noticeable when working with higher voltages are significant in microvolt signals. The Model 3706A reads only the signal received at its input; therefore, it is important that this signal be properly transmitted from the source. The following paragraphs indicate factors that affect accuracy, including stray signal pick-up and thermal offsets.

Shielding

AC voltages that are extremely large compared with the DC signal to be measured may produce an erroneous output. Therefore, to minimize AC interference, the circuit should be shielded, with the shield connected to the Model 3706A input low (particularly for low level sources). Improper shielding can cause the Model 3706A to behave in one or more of the following ways:

- Unexpected offset voltages
- Inconsistent readings between ranges
- Sudden shifts in reading

To minimize pick-up, keep the voltage source and the Model 3706A 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. Therefore, minimize the loop area of the input leads and connect each signal at only one point.

Thermal EMFs

Thermal EMFs (thermoelectric potentials) are generated by temperature differences between the junctions of dissimilar metals. These can be large compared to the signals that the Model 3706A can measure. Thermal 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 touching the circuit, by placing a heat source near the circuit, or by a regular pattern of instability (corresponding to changes in sunlight or the activation of heating and air conditioning systems).

To minimize the drift caused by thermal EMFs, use copper leads to connect the circuit to the Model 3706A.

A clean, oxidized-free, copper conductor such as #10 bus wire is ideal. For switching modules, use #20 AWG copper wire to make connections. The leads to the Model 3706A may be shielded or unshielded, as necessary.

Widely varying temperatures within the circuit can also create thermal EMFs. Therefore, maintain constant temperatures to minimize these thermal EMFs. A shielded enclosure around the circuit under test also helps by minimizing air currents.

The relative offset control can be used to null out constant offset voltages.

AC voltage offset

The Model 3706A, at 5½ digits resolution, will typically display 100 counts of offset on AC volts with the input shorted. This offset is caused by the offset of the T_{RMS} converter. This offset will not affect reading accuracy and should not be zeroed out using the relative offset feature. The following equation expresses how this offset (V_{OFFSET}) is added to the signal input (V_{IN}):

$$\textit{Displayed reading} = \sqrt{(V_{IN})^2 + (V_{OFFSET})^2}$$

Example:
Range= 1 VAC, Offset = 100 counts (1.0 mV), Input = 100 mV_{RMS}

$$\textit{Displayed reading} = \sqrt{(100\text{mV})^2 + (1.0\text{mV})^2}$$

Therefore, the displayed reading is 0.100005 V.

The offset is seen as the last digit, which is not displayed. Therefore, the offset is negligible. If relative offset were used to zero the display, the 100 counts of offset would be subtracted from V_{IN}, resulting in an error of 100 counts in the displayed reading.

Resistance

Optimizing low ohm measurement and speed

When measuring resistance of 100 ohms or less, cable, connectors, and Model 3706A switch cards can have thermal offsets, which can result in additional reading uncertainties.

Auto delays for 100 ohms or less have been optimized for throughput and settling, resulting in the best measurement. If thermal offsets cause additional uncertainty, adding a delay of 10 ms can improve accuracies. Refer to the Auto Delay table for additional details. If the application requires additional settling delay, send the following commands to the channel or slot of interest:

```
channel.setdelay("4004", 0.010)
```

Adds 10 ms of delay after closing channel 4 in slot 4.

```
channel.setdelay("slot4", 0.010)
```

Adds 10 ms of delay to all channels in slot 4.

Dry circuit ohms (DRY+)

Standard resistance measurements have open-circuit voltage levels from 6.4 V to 14.7 V, depending on the selected range. Dry circuit ohms limits open-circuit voltage to between 20 mV and 27 mV. This allows you to perform resistance measurements that require low open-circuit voltage, such as power and low-glitch resistance measurements.

Dry circuit ohms can be used on the 1 Ω, 10 Ω, 100 Ω, 1 kΩ, and 10 kΩ ranges (maximum resistance of 2.4 kΩ) for the four-wire ohm function.

Offset-compensated ohms (OC+) can be used with dry circuit ohms to cancel the effect of thermal EMFs. When dry circuit is enabled, offset compensation is automatically set to on.

Measuring contact resistance (oxide film build-up)

The ideal resistance between switch connectors, or relay contacts is 0 Ω. However, an oxide film may be present on the switch or relay contacts. This oxide film could add resistance on the order of several hundred milliohms. Also, this oxide film changes the contact resistance over time and with changes in the environmental conditions (such as temperature and humidity).

Typically, the four-wire ohm function of the Model 3706A or a standard DMM is used to measure low resistance. However, if standard resistance measurements are performed, the relatively high open-circuit voltage may puncture the oxide film, and render the test meaningless.

Dry circuit ohms limit voltage to 20 mV to minimize any physical and electrical changes in a measured contact junction. This low open-circuit voltage will not puncture the film, and will therefore provide a resistance measurement that includes the resistance of the oxide film.

Oxide films may also build up in connections on a semiconductor wafer. To accurately measure the resistance introduced by the oxide film, dry circuit ohms should be used to prevent oxide film puncture.

Measuring resistance of voltage-sensitive devices

Dry circuit ohms should be used for any device that could be damaged by high open circuit voltage. If you are not sure the slightly degraded accuracy is a consideration, it is good practice to use dry circuit ohms to measure low resistance.

Dry circuit ohms measurement considerations

Dry circuit ohms uses a constant current source with voltage monitoring that is used to clamp the current source voltage. The current source will remain constant as long as the monitoring voltage is less than 20 mV. When the voltage exceeds 20 mV, the current source shunts current internal to the DMM until 20 mV is maintained at the DUT.

When using dry circuit ohms, the DUT is shunted by 100 kΩ and 0.9 μF for the 1, 10, and 100 ohm ranges. For the 1 K and 10 K ranges, it is shunted by a 10 MΩ and 0.015 μF. This allows the current source to have minimal overshoot voltage under transient conditions. When used with a switching system, the overshoot is less than 40 mV in 20 μs.

Enable or disable dry circuit ohms from the front panel

Dry circuit ohms is an attribute set for the 4-wire ohm DMM function.

NOTE

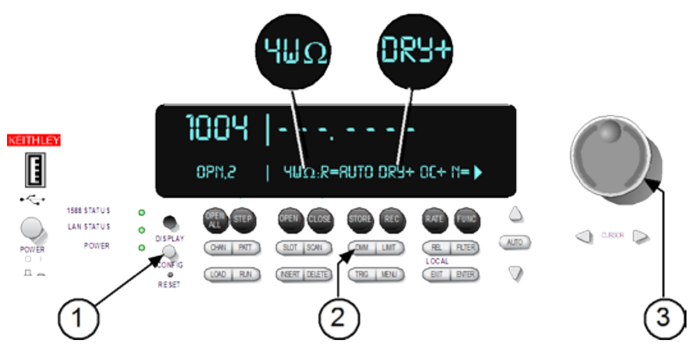
When the dry circuit ohms attribute is enabled, the offset-compensated ohms attribute is automatically enabled (OC+ annunciator). If you do not wish to use offset-compensated ohms, after setting dry circuit ohms, disable offset-compensated ohms using the information in Enabling/disabling offset-compensated ohms (on page 4-62).

NOTE

If the Model 3706A is being controlled remotely, press the **EXIT** key to place it in local mode.

To enable offset-compensated ohms, the Model 3706A DMM function must be set to four-wire ohms. The Model 3706A is in 4-wire ohm mode when **4WΩ** is displayed. Dry Circuit is active when the DRY+ is displayed (see the figure below).

Figure 85: Enabling dry-circuit ohms



To enable/disable dry circuit ohms from the front panel:

1. Press the **CONFIG** key ①.
2. Press the **DMM** key ②.
3. Turn the navigation wheel ③ to scroll to the "DRYCIRCUIT" menu item.
4. Press the navigation wheel ③ to display ON/OFF settings for dry circuit ohms.
5. Select "ON" or "OFF" and press the navigation wheel ③ again.
6. Press the **EXIT** key to leave the menu.

Figure 86: Four-wire Ohm ATTR MENU: DRYCIRCUIT

NOTE

When enabled, the dry circuit ohms annunciator is on (DRY+).

Enable or disable dry circuit ohms through the remote interface

To enable dry circuit ohms through the remote interface, send the commands:

```
dmm.func = "fourwireohms"
dmm.drycircuit = dmm.ON
```

To disable dry circuit ohms through the remote interface, send the command:

```
dmm.drycircuit = dmm.OFF
```

Performing dry circuit ohms measurements

Make sure you use four-wire connections to the DUT as detailed in Analog backplane connector (rear panel) (on page 4-19) or specific to the module used for switching.

To perform dry circuit ohms measurements:

NOTE

Do not make connections to the device under test (DUT) until after the dry circuit ohms attribute is enabled in step 2.

1. Press the **OPENALL** key to open all switching channels.
2. If not already on, enable dry circuit ohms (see Enabling/disabling dry circuit ohms (see "Enable or disable dry circuit ohms from the front panel" on page 4-60)).
   - Dry circuit ohms enabled: DRY+
   - Dry circuit ohms disabled: DRY-

NOTE

When dry circuit measurement is enabled (DRY+), offset-compensated ohms will also enable (OC+ annunciator turns on). If you do not wish to use offset-compensated ohms, disable it (see Enabling/disabling offset-compensated ohms (on page 4-62)).

1. Make 4-wire connections to the DUT. See 4-wire connection information contained in analog backplane connector (rear panel) (on page 4-19) and Switching module (see "Switching module connections" on page 4-20).
2. Use the **RANGE** ⋏ and ⋎ keys to select the 1Ω, 10Ω, 100Ω, 1kΩ, or 10kΩ range, or press the **AUTO** key to enable auto range.
3. If using a switching module, perform the following steps to close the desired channel:
   a. Use the navigation wheel to dial in the channel number.
   b. Press the **CLOSE** key.

1. Press the **TRIG** key and observe the displayed reading. If the "Overflow" message is displayed, select a higher range until a normal reading is displayed (or press the **AUTO** key for autoranging). For manual ranging, use the lowest possible range for the best resolution.
2. To measure other switching channels, repeat steps 5 and 6.
3. When finished, press the **OPENALL** key to open all channels.

NOTE

The on or off states of dry circuit ohms and offset-compensated ohms are saved with four-wire ohm function. If you select a different measurement function, then select four-wire ohms again, the previous attribute states of dry circuit ohms and offset-compensated ohms will be restored.

Offset-compensated ohms

The presence of thermal EMFs (V_{EMF}) can adversely affect low-resistance measurement accuracy. To overcome these unwanted offset voltages, you can use offset-compensated ohms. Offset-compensated ohms measurements can be performed on the 1 Ω, 10 Ω, 100 Ω, 1 kΩ, and 10 kΩ ranges for the four-wire ohm measurement function. It cannot be done on the two-wire ohm measurement function.

NOTE

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

For a normal resistance measurement, the Model 3706A sources a current (I) and measures the voltage (V). The resistance (R) is then calculated as (R=V/I) and the reading is displayed.

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

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

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

where:

$$\Delta V = V2 - V1$$

$$\Delta I = I2 - I1$$

V1 is the voltage measurement with the current source at its normal level.

V2 is the voltage measurement using the lowest current source setting.

I1 is the current measurement with the source set to a specific level.

I2 is the current measurement with the source set to zero.

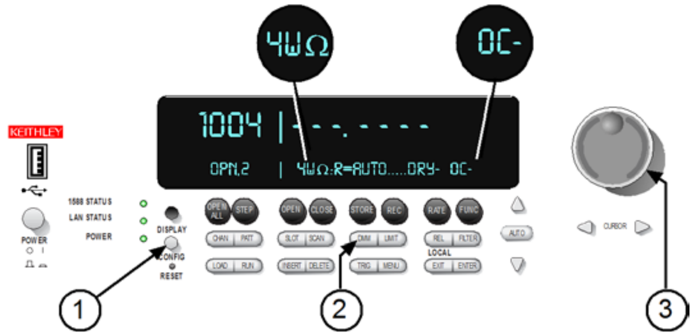
This 2-point measurement process and reading calculation eliminates the resistance contributed by the presence of V_{EMF}.

When the source is turned on, the output cycles between the programmed value and zero (0 A or 0 V) to derive the offset-compensated ohms measurement.

Enabling/disabling offset-compensated ohms

Offset-compensated ohms is an attribute that can be set for the 4-wire ohms measurement function. To enable or disable it from the front panel:

Figure 87: Enabling offset-compensated ohms



NOTE

The Model 3706A is in 4-wire ohm mode when 4WΩ is displayed. Offset compensation is active when the OC+ is displayed (OC- is shown in the above figure).

1. Set the Model 3706A for the 4-wire ohm measurement function.
2. Press the **CONFIG** key ①.
3. Press the **DMM** key ②.
4. Turn the navigation wheel ③ to scroll to the "OFFSETCOMP" menu item. Press the navigation wheel to select.
5. Turn the navigation wheel to select the ON/OFF settings for Offset Compensation as desired and press the navigation wheel to set.
6. Press the **EXIT** key to leave the menu.

Figure 88: Four-wire Ohm ATTR MENU: OFFSETCOMP

Performing offset-compensated ohms measurements

Make sure you use 4-wire connections to the DUT as detailed in analog backplane connector (rear panel) (on page 4-19) or if using a module for switching, the connections specific to the module.

1. Press the **OPENALL** key to open all switching channels.
2. If not already on, enable offset compensated ohms (OC+ annunciator is lit). See Enabling/disabling offset-compensated ohms (on page 4-62).
3. Use the **RANGE** ⮝ and ⮟ keys to select the range, or press the **AUTO** key to enable auto range. If using auto range, offset-compensated ohms measurements will not be performed if the instrument goes to the 100 kΩ (or higher) range.
4. Perform steps 4 through 8 of the Standard resistance measurements (see "Resistance measurements from the front panel" on page 4-22) procedure.

Offset compensation can be enabled for any 4-wire range. The internal DMM will perform offset compensation for the <=10k-ohm ranges and automatically disable for >=100K-ohm ranges. Send the following remote commands and the Series 3700A returns the reading and a "1" or a logic "True", but the DMM only performed a standard 4-wire measurement.

```
dmm.func="fourwireohms"
dmm.range=100e3
dmm.offsetcompensation=1
print(dmm.measure())
print(dmm.offsetcompensation)
```

With dry circuit ohms enabled, the 10 kΩ range (measuring a maximum resistance of 2.4 kΩ) is the highest offset-compensated ohms range that can be selected.

For buffer recall, there is no way to distinguish between a normal ohms reading and an offset-compensated ohms reading. The OC annunciator (– or +) has no significance for recalled resistance readings that are displayed.

The offset-compensated ohms setting is saved with the measurement function. If you change measurement functions, then return to the previous function, the offset-compensated ohms will be at the same setting it was previously.

`dmm.offsetcompensation` is a common command and is shared with `fourwireohms`, `drycircuit`, `threertd` and `fourrtd`. To activate `dmm.offsetcompensation`, select the desired function first, and then send `dmm.offsetcompensation = dmm.ON` or `OFF`.

Filter

You can use the digital filter to stabilize noisy measurements. When the filter is applied, the displayed, stored, or transmitted reading is a windowed-average of a number of reading conversions (from 1 to 100).

The filter setup is saved specific to each measurement function (DC volts, AC volts, DC current, AC current, two-wire ohms, four-wire ohms, commonside ohms, and temperature). When you select a function, the instrument will return to the last filter that was set up for that function.

NOTE

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

Filter characteristics

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

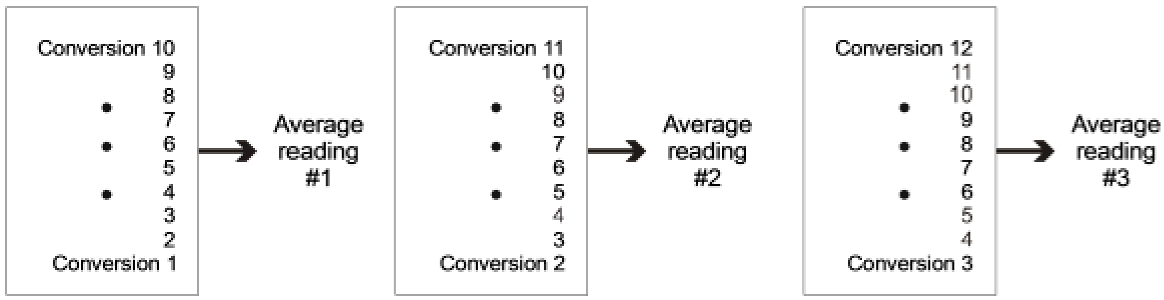
Digital filter types

There are two digital filter types: Moving and repeating.

Moving average filter

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

Figure 89: Moving average filter

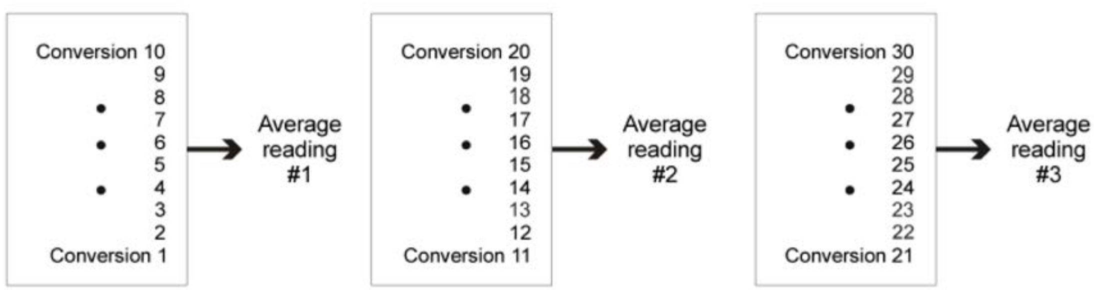


Repeating average filter

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

> **NOTE**
>
> The moving filter cannot be used when scanning (see Scanning). If a scan channel is set up to use the moving filter, the filter will not turn on.

Figure 90: Repeating average filter



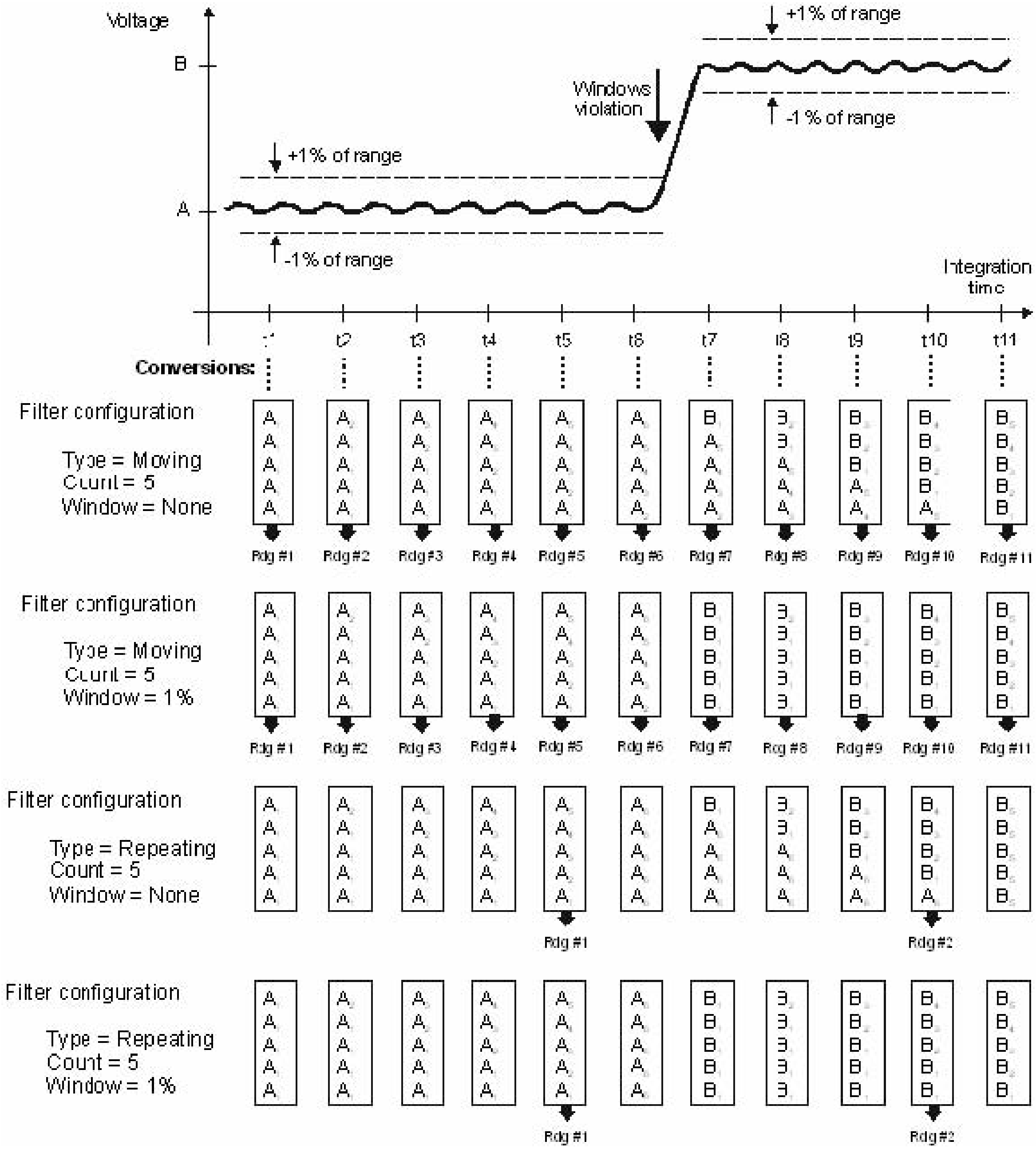
Digital filter window

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

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

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

Figure 91: Filter window



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

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

For temperature, the filter window is expressed as a percent of the maximum temperature reading. The maximum temperature depends on which thermocouple is being used. For example, for a Type J thermocouple, the maximum reading is 760 °C; a 10 % window means that the filter window is ±76 °C. For temperatures below 0 °C, the overflow point is –200 ºC, so a 10% filter window is ±20 ºC. If using ºF units, a 20% filter window is calculated as follows: 9/5 x 20 = 36. The filter window for the 20% window is ±36 ºC.

Section 5

Theory of operation

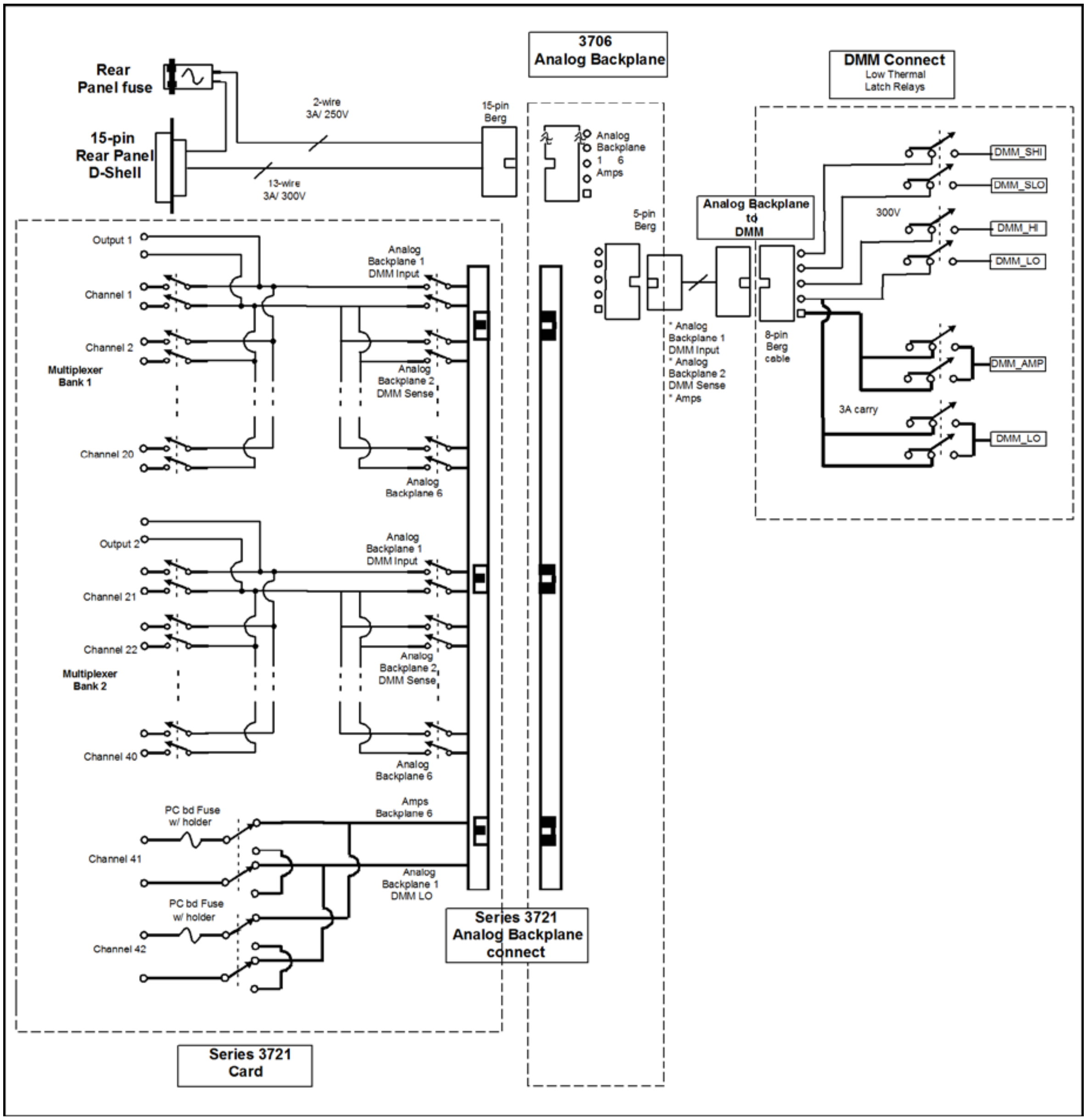
In this section:

DMM

Rear panel, backplane, and DMM connect relays schematic

Refer to the following figure for a schematic of the rear panel, backplane, and DMM connect relays with a typical card.

Figure 92: Rear panel to backplane to DMM connect relays schematic



Line cycle synchronization

Synchronizing A/D conversions with the frequency of the power line increases common mode and normal mode noise rejection. When line cycle synchronization is enabled, the measurement is initiated at the first positive-going zero crossing of the power line cycle after the trigger.

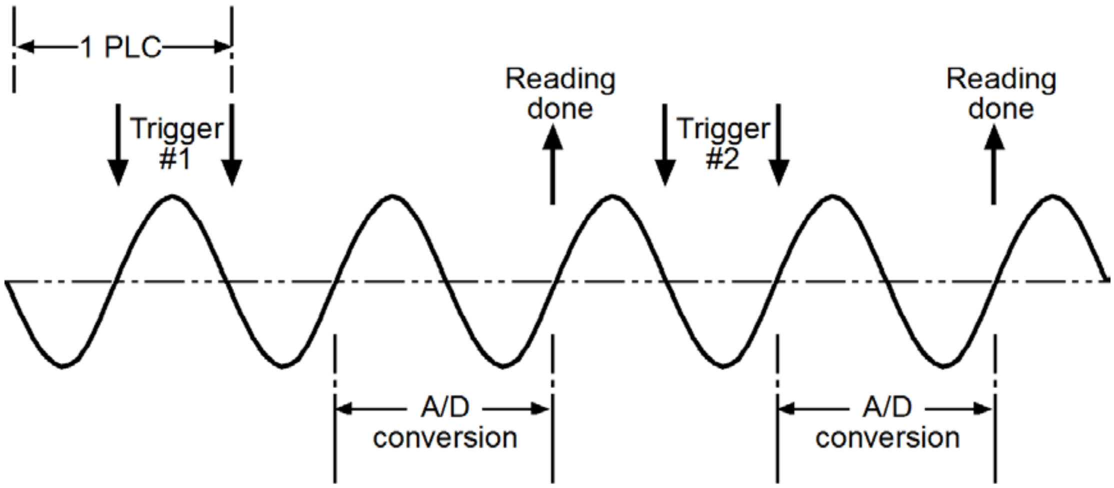
The following figure shows a measurement process that consists of two A/D conversions. If the trigger occurs during the positive cycle of the power line (Trigger #1), the A/D conversion starts with the positive-going zero crossing of the power line cycle. If the next trigger (Trigger #2) occurs during the negative cycle, then the measurement process also starts with the positive-going zero crossing.

NOTE

Line synchronization is not available for the AC functions (ACV, ACI, FREQ, or PERIOD). Line synchronization can be enabled for any DC function and any NPLC measurement, increasing NMRR and CMRR.

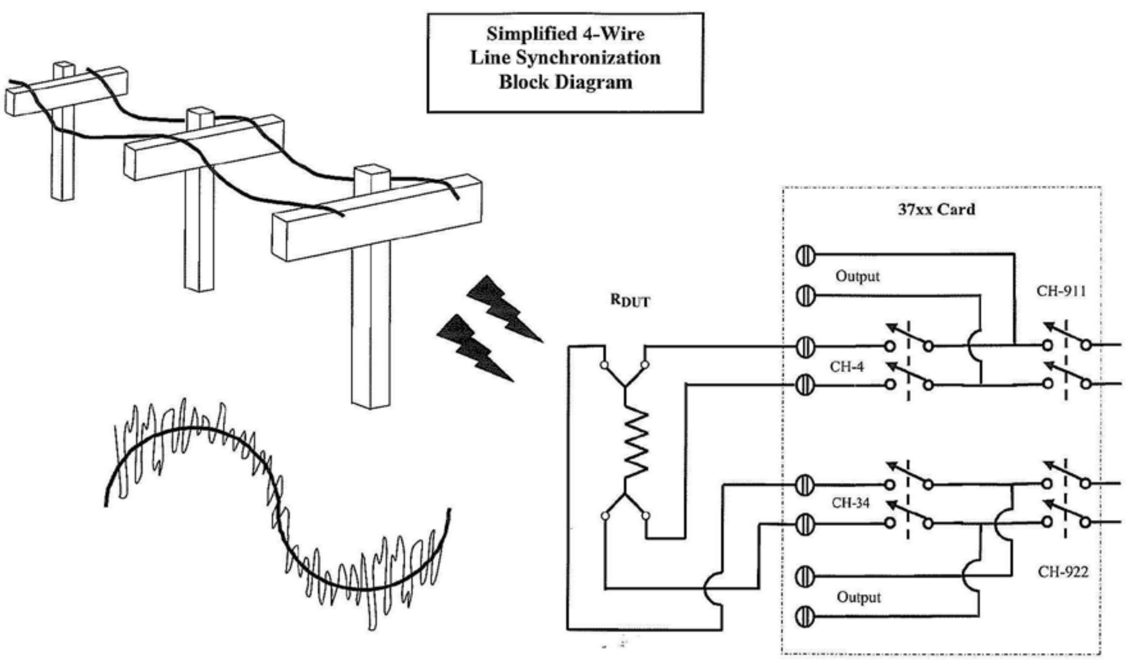
See dmm.linesync (on page 8-203) in the Reference Manual for remote programming information.

Figure 93: Line cycle synchronization



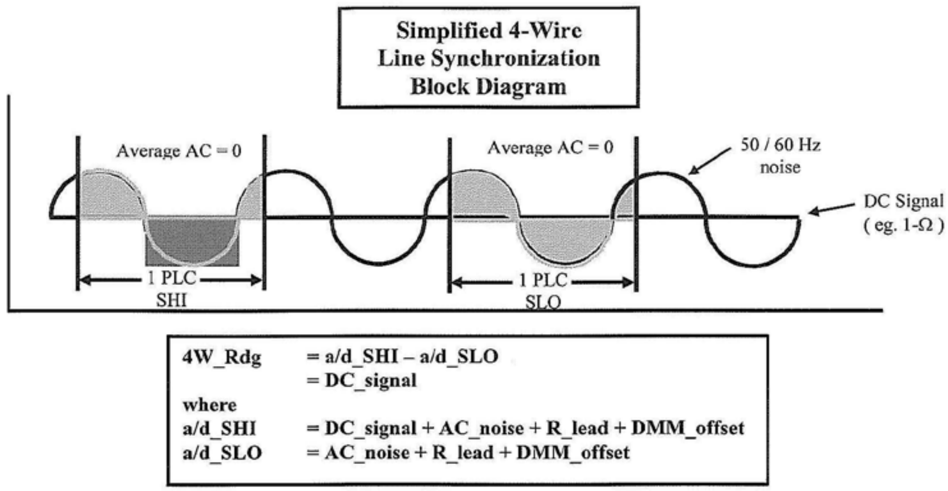
4-Wire Ohms are sensitive to 50 / 60Hz power line noise, due to cabling and Model 3700A switch card loop area.

Figure 94: 4-Wire Line Synchronization Block Diagram



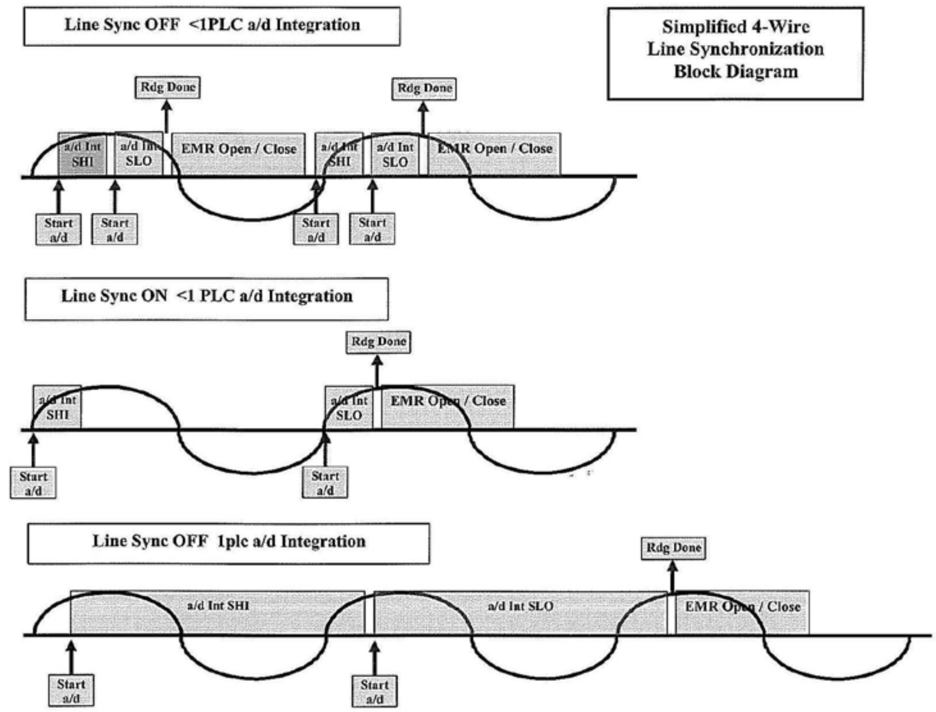
Traditional DMM 4-wire measurements are made in two phases, S HI and S LO. If the `dmm.nplc` is set to 1 or a multiple of the power line, theoretically, the average AC noise is 0. Refer to the 4W-Rdg calculation (http://www.maxim-ic.com/appnotes.cfm/appnote_number/1041/).

Figure 95: 1plc Line Sychronization Block Diagram



For Line Synchronization off and <1plc, reading rate increases, but measurement uncertainty and noise increases due to the Average AC noise during the S HI phase not canceling with the S LO phase. With line synchronization ON, the S HI and S LO measurement phases are triggered at the rising edge of the power line zero crossing. This improves reading uncertainty and noise by >30x while minimal reading rate reduction.

Figure 96: Line Sync Off and On <1plc



AC voltage measurements and crest factor

The root-mean-square (RMS) value of any periodic voltage or current is equal to the value of the DC voltage or current which delivers the same power to a resistance as the periodic waveform does. Crest factor is the ratio of the peak value to the RMS value of a particular waveform. This is represented by the following equations:

$$CF = \frac{V_P}{V_{RMS}} \quad \text{or} \quad CF = \frac{I_P}{I_{RMS}}$$

The crest factor of various waveforms is different, because the peak-to-RMS ratios are variable. For example, the crest factor for a pulse waveform is related to the duty cycle; as the duty cycle decreases, the crest factor increases. The RMS calculations and crest factor (CF) for various waveforms are shown in the following figures.

Figure 97: RMS calculations and crest factor

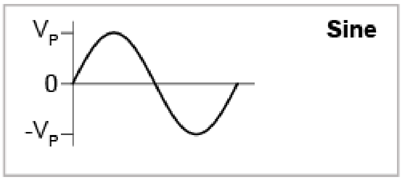
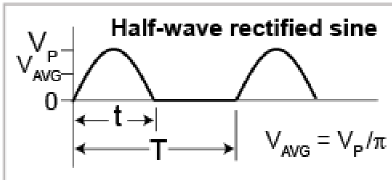
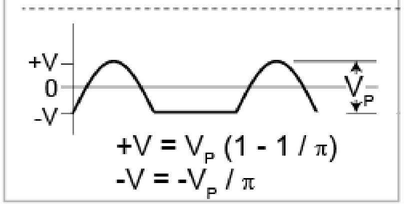
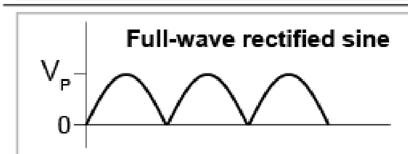
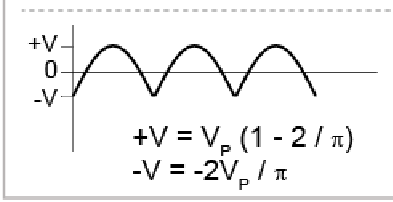
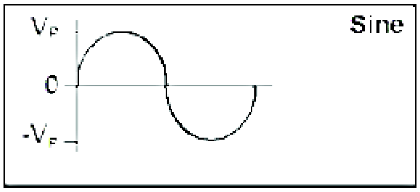
Waveform	RMS	Crest factor
	**AC coupled RMS:** $V_{RMS} = \frac{V_P}{\sqrt{2}}$	**Crest factor:** $CF = \sqrt{2}$
	**RMS:** $V_{RMS} = V_P\sqrt{D/2}$ where: D (duty cycle) = t / T	**Crest factor:** $CF = \frac{1}{\sqrt{D/2}}$
	**AC coupled RMS:** $V_{RMS} = \sqrt{(V_P\sqrt{D/2})^2 - (V_P/\pi)^2} = V_P\sqrt{(D/2) - (1/\pi^2)}$	**Crest factor:** $CF = \frac{1}{\sqrt{(D/2) - (1/\pi^2)}}$
	**RMS:** $V_{RMS} = \frac{V_P}{\sqrt{2}}$	**Crest factor:** $CF = \sqrt{2}$
	**AC coupled RMS:** $V_{RMS} = \sqrt{(V_P/\sqrt{2})^2 - (2V_P/\pi)^2}$ $V_P\sqrt{(1/2) - (4/\pi^2)}$	**Crest factor:** $CF = \frac{1}{\sqrt{(1/2) - (4/\pi^2)}}$

Figure 98: RMS calculations and crest factor

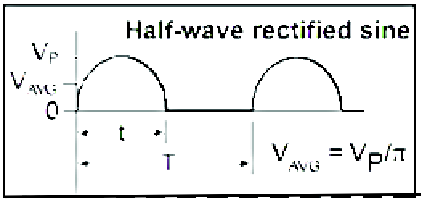


AC coupled RMS:

$$V_{RMS} = \frac{V_P}{\sqrt{2}}$$

Crest Factor:

$$CF = \sqrt{2}$$



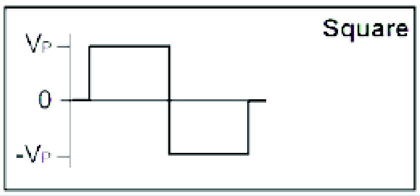
RMS:

$$V_{RMS} = V_P\sqrt{D/2}$$

where. D (duty cycle) = t/T

Crest Factor:

$$CF = \frac{1}{\sqrt{D/2}}$$

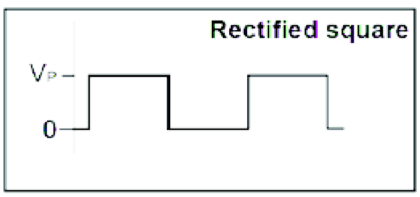


AC coupled RMS:

$$V_{RMS} = V_P$$

Crest Factor:

$$CF = 1$$

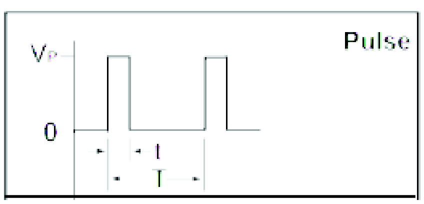


AC coupled RMS:

$$V_{RMS} = \frac{V_P}{2}$$

Crest Factor:

$$CF = 2$$

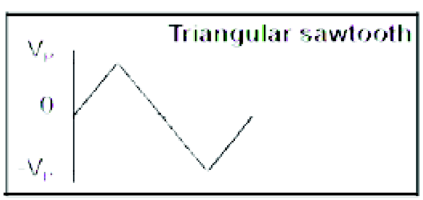


AC coupled RMS:

$$\frac{V}{CF} \times \sqrt{1 - \frac{1}{CF^2}}$$

Crest Factor:

$$\sqrt{\frac{T}{t}}$$



RMS:

$$V_{RMS} = 0.557V_P$$

Crest Factor:

$$CF = 1.733$$

The Model 3706A is an AC-coupled RMS meter. For an AC waveform with DC content, the DC component is removed before the RMS is calculated. This affects the crest factor because the peak value for the DC-coupled waveform is different than the peak value for the AC-coupled waveform. In an AC-coupled waveform, the peak value is measured from the original DC average value, not DC zero. For example, if a voltage pulse is measured on the AC function of the Model 3706A with a peak voltage of VP and a low voltage of zero volts, the AC-coupled peak value will be calculated as follows:

ACPEAK = VP • (1 - duty cycle)

Therefore, the AC-coupled crest factor will differ from the DC-coupled waveform. The RMS function will calculate the RMS value based on the pulsed waveform with an average value of zero.

The reason to consider crest factor in accuracy of RMS measurements is because the meter has a limited bandwidth. Theoretically, a sine wave can be measured with a finite bandwidth because all of its energy is contained in a single frequency. Most other common waveforms have a number of spectral components requiring an almost infinite bandwidth above the fundamental frequency to measure the signal exactly. Because the amount of energy contained in the harmonics becomes smaller with increasing frequency, very accurate measurements can be made with a limited bandwidth meter, as long as enough spectral components are captured to produce an acceptable error.

Crest factor is a relative measurement of the harmonic content of a particular waveform and reflects the accuracy of the measurement. For a rectangular pulse train, the higher the crest factor, the higher the harmonic content of the waveform. This is not always true when making spectral comparisons between different types of waveforms. A sine wave, for example, has a crest factor of 1.414, and a square wave has a crest factor of 1. The sine wave has a single spectral component and the square wave has components at all odd harmonics of the fundamental.

The Model 3706A RMS AC volts and AC amps accuracies are specified for sine waves of different frequency ranges.

Additional error uncertainties are also specified for non-sinusoidal waveforms of specific crest factors and frequencies. The Model 3706A has capabilities of measuring AC waveforms of crest factors up to 5.

DMM resistance measurement methods

The method that the Model 3706A uses to measure resistance depends on the resistance range. For resistance ranges from 1 Ω to 1 MΩ , the Model 3706A uses the constant-current method to measure resistance. For resistance ranges from 10 MΩ to 100 MΩ ranges, the ratiometric method is used.

When the constant-current method is used, the Model 3706A 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 6.4 V reference through a 10 MΩ reference resistance (R_{REF}).

For more detail on these methods, see Constant-current source method (on page 5-9) and Ratiometric method (on page 5-9).

The Model 3706A uses four methods to detect open leads. For detail, see Open lead detection (on page 5-14).

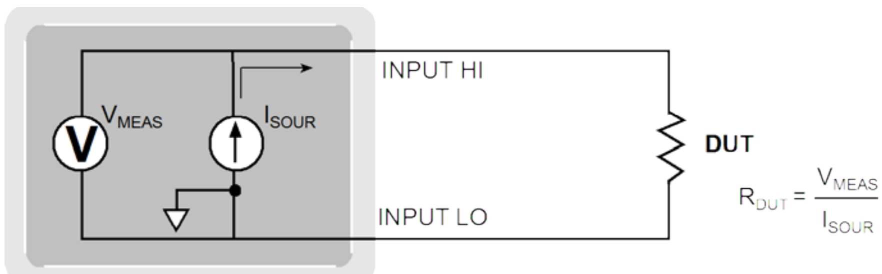
Constant-current source method

For the 1 Ω to 1 MΩ ranges, the Model 3706A uses the constant-current method to measure resistance. The Model 3706A 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 ($R_{DUT} = V_{MEAS}/I_{SOUR}$).

The constant-current method is shown below. 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 Model 3706A has very high input impedance (>10 GΩ), virtually all the test current (1 mA) flows through the DUT. For DUT ≤1 kΩ, 4-wire ohms measurements should be used as shown. Because the voltage is measured at the DUT, voltage drop in the test leads is eliminated (this voltage could be significant when measuring low-ohm DUT).

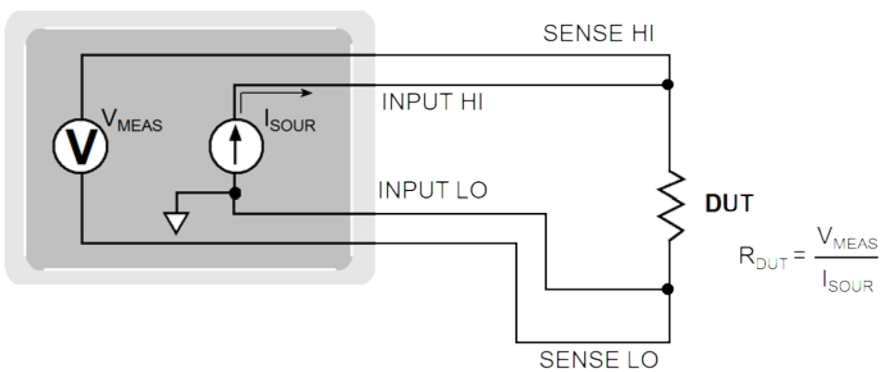
The 2-wire constant-current method is shown below.

Figure 99: Two-wire constant-current source method



The 4-wire constant-current method is shown below.

Figure 100: Four-wire constant-current source method

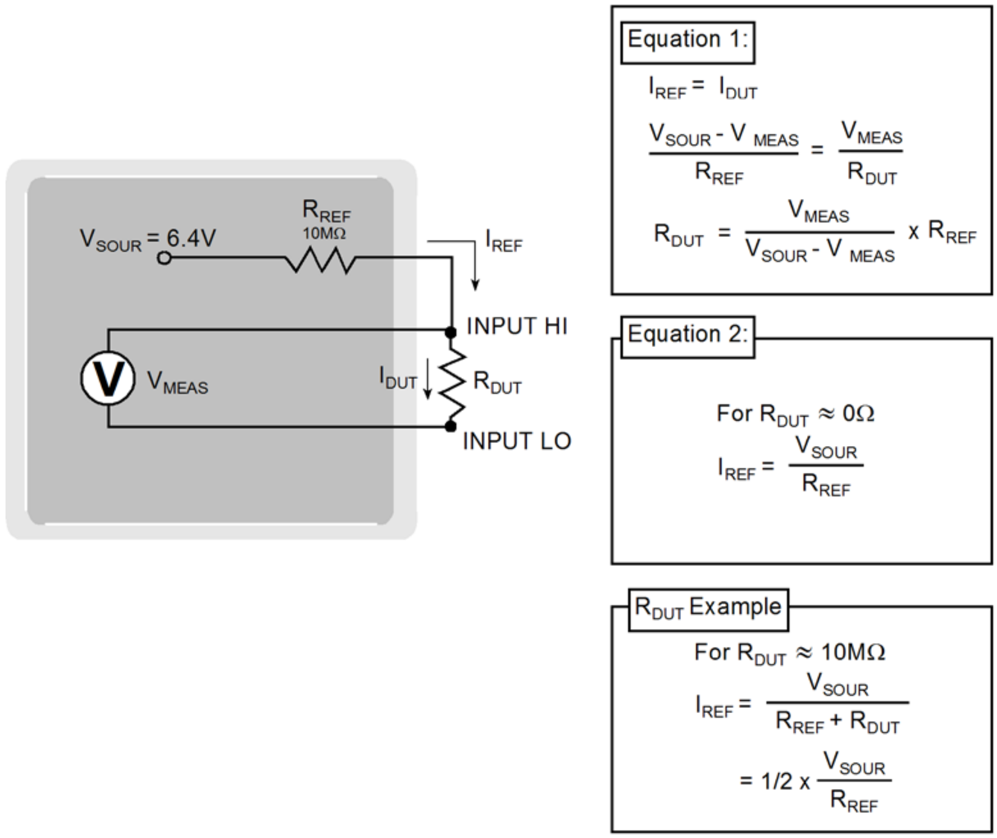


Ratiometric method

For the 10 MΩ and 100 MΩ ranges, the ratiometric method is used to measure resistance. Test current for this method is generated by a 6.4V voltage source through a 10 MΩ reference resistance (R_{REF}), as shown.

Basic circuit theory dictates that I_{REF} is equal to the I_{DUT}. Because the voltmeter of the Model 3706A (V_{MEAS}) has very high input impedance (>10GΩ), current through the voltmeter branch is insignificant and can be discounted. Therefore, as shown in the following Figures Equation 1, $I_{REF} = I_{DUT}$

Figure 101: Two-wire ratiometric method



Because I = V/R, Equation 1 is modified using the V/R equivalents in place of I_{REF} and I_{DUT}. Therefore:

$I_{SOUR} = (V_{MEAS} / R_{REF}) + (V_{MEAS} / R_{DUT})$

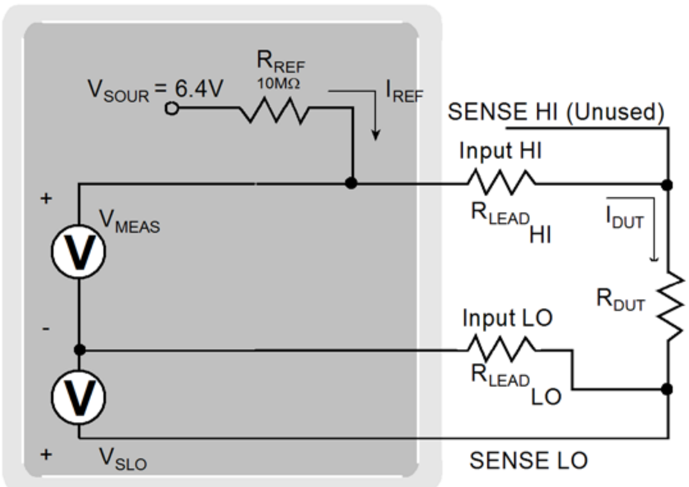
Note that V_{MEAS} is measured by the Model 3706A. With V_{MEAS}, I_{SOUR}, R_{REF} known, the Model 3706A calculates the resistance of the DUT and displays the result. R_{REF} is learned during calibration and V_{SOUR} is routinely self-calibrated when the `dmm.autozero` attribute is enabled (`dmm.autozero = dmm.ON`).

As shown, the four-wire ohm function can also be used to measure ohms for the 10MΩ and 100 MΩ ranges. To minimize the effects of charge injection when dmm-autozero is enabled, the 10 MΩ to 100 MΩ is actually a 3-wire ohm measurement. SENSE HI is not used. SENSE HI is connected to the DUT but is not required (it can be left open). The measurement method is similar to the ratiometric method for two-wire ohms, but it performs an extra voltage measurement (V_{LEAD}) to compensate for voltage drop in the input test leads.

Note that 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 x V_{LEAD}. Therefore:

$V_{DUT} = V_{MEAS} - 2(V_{LEAD})$.

Figure 102: Four-wire ratiometric method



Equation

$$I_{REF} = I_{DUT}$$

$$= \frac{V_{MEAS}}{R_{LEAD_{HI}} + R_{DUT} + R_{LEAD_{LO}}}$$

Assume:

$$R_{LEAD_{HI}} = R_{LEAD_{LO}}$$

Assume:

$$V_{SLO} = R_{LEAD_{LO}} \times I_{REF}$$

$$\therefore R_{LEAD_{LO}} = \frac{V_{SLO}}{I_{REF}}$$

Substitute:

$$I_{REF} = \frac{V_{MEAS}}{2\,(R_{LEAD_{LO}}) + R_{DUT}}$$

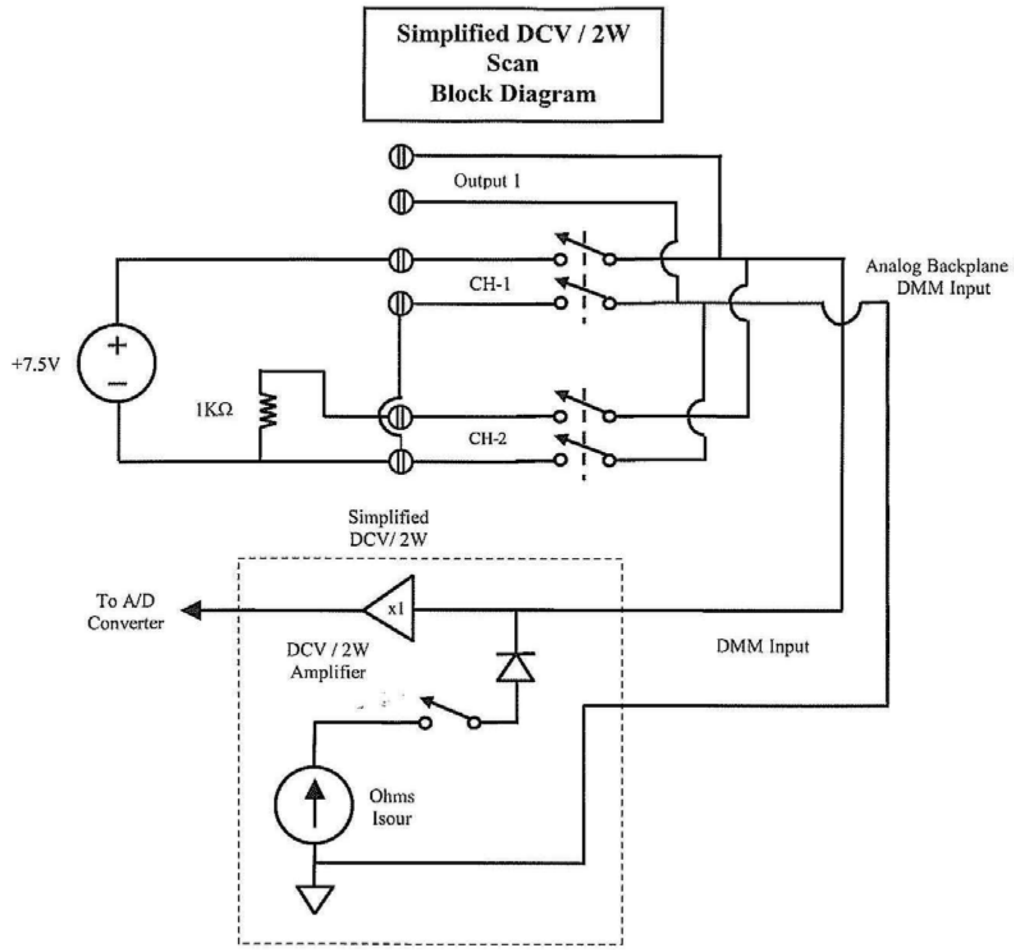
$$= \frac{V_{MEAS}}{2\,(V_{SLO}/I_{REF}) + R_{DUT}}$$

$$= \frac{V_{MEAS} - 2V_{SLO}}{R_{DUT}}$$

$$R_{DUT} = \frac{V_{MEAS} - 2V_{SLO}}{I_{REF}}$$

$$= \frac{V_{MEAS} - 2V_{SLO}}{V_{SOUR} - V_{MEAS}} \times R_{REF}$$

Figure 103: Fast Alternating Scan block diagram



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 Model 3706A 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.

There are two ways for the Model 3706A to acquire the cold junction temperature. It can measure the cold junction using a thermistor or 4-wire RTD, or the known temperature value can be entered by the user.

There are two reference junction types supported by the Model 3706A:

- Simulated reference junction
- Internal reference junction
- External reference junction

These reference junctions are explained in the following paragraphs.

Simulated reference junction

An example of a simulated reference junction is an ice bath as shown in the paragraph titled Thermocouple connections (on page 4-26). The copper 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 Model 3706A. The simulated reference temperature for the Model 3706A can be set from 0 °C to 65 °C.

The Model 3706A measures the input voltage and factors in the simulated reference temperature to calculate the temperature reading at the thermocouple.

NOTE

The most accurate temperature measurements are achieved by using a simulated reference junction using an ice point reference.

Internal reference junction

"Internal" implies that temperature transducers are used to measure the cold junction. 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 (see Thermocouple connections (on page 4-26)).

The Model 3706A measures the temperature of the cold junction (screw terminals), measures the input voltage, and then calculates the temperature reading at the thermocouple.

To help maintain stability and accuracy over time and changes in temperature, the Model 3706A 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. Note that internal temperature references are collected regardless of whether or not autozero is enabled.

External reference junction

Thermocouple readings may be configured to use an external reference junction setting. The Series 3700 assumes the external reference junction is connected to channel 1 of a slot. It is recommend that this channel be configured for thermistor or RTD temperature reading. However, the unit does not error check against this. Each time a reading is taken on the external reference junction channel (channel 1 of a slot) it will be used as the new external reference junction value in subsequent external reference readings. External reference readings work with `dmm.close` as well as scanning.

For non-simulated thermocouple measurements, first perform a thermistor or RTD measurement prior to enabling external reference junction.

Open lead detection

The Model 3706A has four methods to detect open lead conditions:

- I_{SOUR} open voltage
- V_{MEAS} open voltage
- Calculated measurement
- `dmm.opendetector`

The following figures show open lead detection schematics for various measurements.

Figure 104: Simplified normal 4-wire ohm open detection schematic

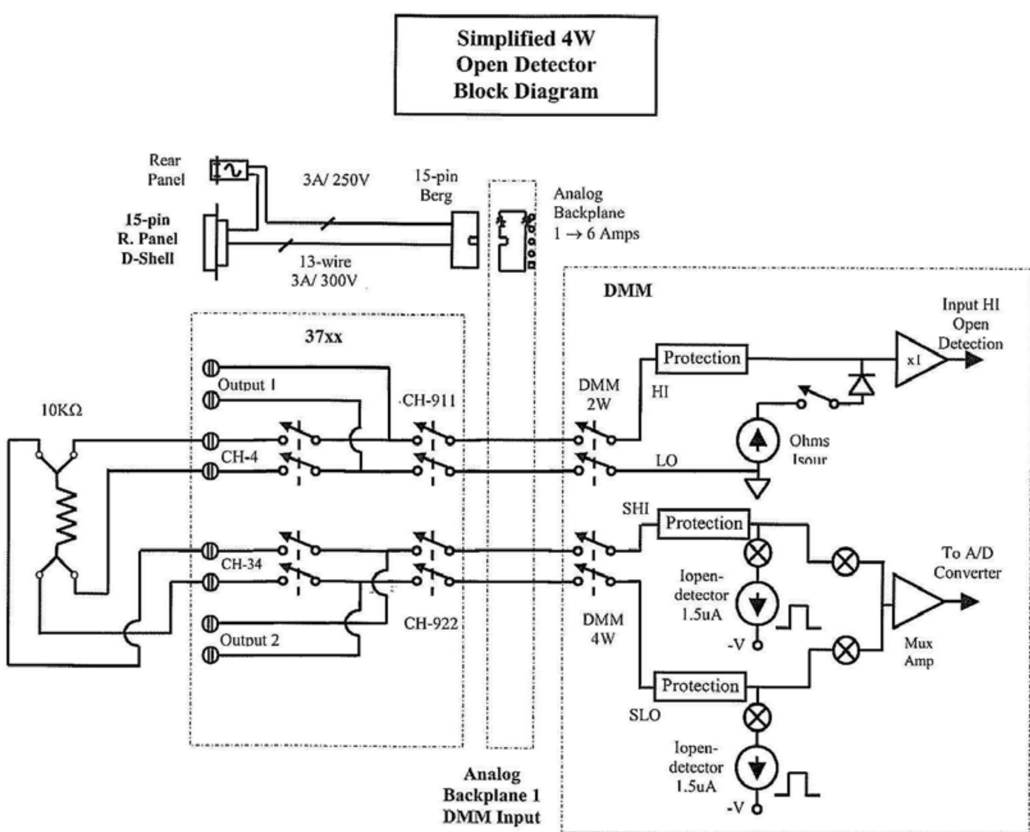
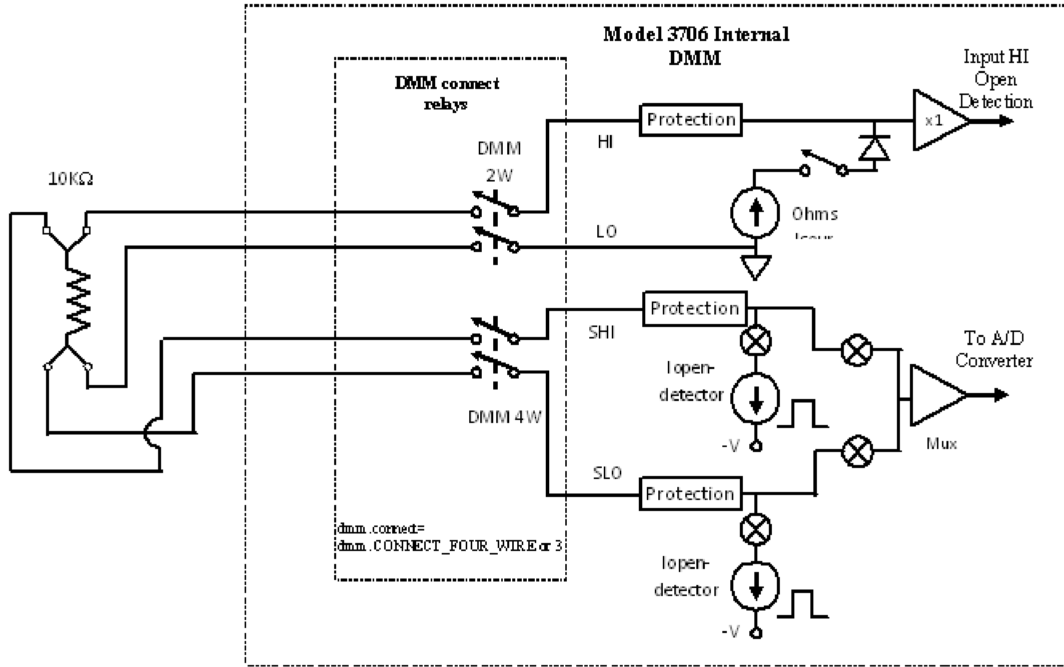


Figure 105: Model 3706A Internal DMM



ISOUR open voltage

1 Ω through 1 MΩ ranges: A hardware detector detects an open input lead. The hardware detector uses a comparator circuit to monitor the voltage on the ohm I_{SOUR} $V_{OPEN\text{-}HI\text{-}LEAD}$ terminal.

- For the lower ohms ranges (1 Ω, 10 Ω, and 10 kΩ), open circuit voltage on the ohm $I_{SOUR\ VOPEN\text{-}HI\text{-}LEAD}$ terminal is more than 7.1 V.
- For the higher ohms ranges (100 kΩ through 1 MΩ), open circuit voltage on the ohm $I_{SOUR\ VOPEN\text{-}HI\text{-}LEAD}$ terminal is more than 12.8 V.

When an input lead (HI or LO) is open, as shown, voltage rises to the open-circuit level, then the A/D will abort in less than 100 µsec and the "Overflow" message is displayed.

VMEAS open voltage

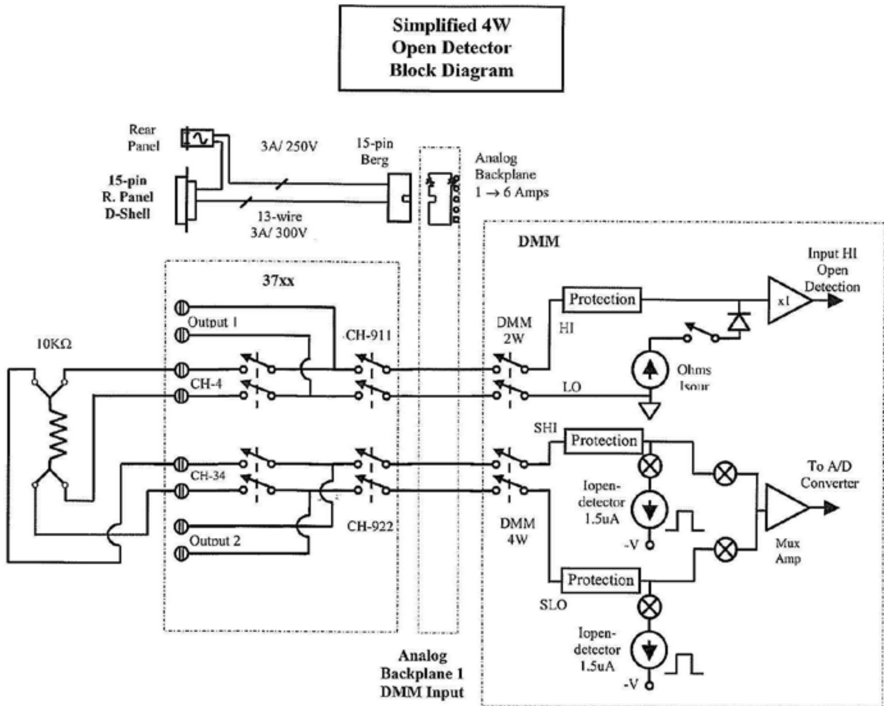
If either Input Sense HI or Sense LO V_{MEAS} is outside the enclosed table voltages, the A/D will stop in less than 100 µsec and return an overflow reading.

Range	V_{MEAS} SHI or SLO high limit open lead detection	V_{MEAS} SHI or SLO low limit open lead detection
1 Ω to 100 Ω	> 128 mV	< –10 mV
1 kΩ to 100 kΩ	> 1.28 V	< –100 mV
1 MΩ	> 12.8 V	< –1.0 V

Calculated measurement open voltage

A calculated measurement that exceeds 120 percent of the range returns an overflow reading.

Figure 106: 4-wire open detector with Series 3700A card



dmm.opendetector open voltage

With `dmm.opendetector = dmm.ON`, a separate -1.5 A $I_{OPENLEAD}$ SHI and a separate SLO current source will pulse on and off before the start of each measurement while I_{SOUR} remains enabled. The A/D will monitor SHI for 2 ms, then switch to SLO for an additional 2 ms. During either phase, if the input voltage exceeds the above table, the A/D will stop in less than 100 µs and return an overflow reading. If there are no open leads detected during the $I_{OPENLEAD}$ phase, the $I_{OPENLEAD}$ is disabled and standard 4-wire is enabled.

V$_{MEAS}$ with open input:

If Sense HI is disconnected, V_{MEAS} will droop less than –1V, causing an A/D overflow.

V$_{MEAS}$ with valid connections:

For valid connections, INPUT Sense HI, V_{MEAS}, will dip during the 4 ms $I_{OPENLEAD}$ phase. The amount of the voltage dip is the sum of $I_{OPENLEAD}$ and the range I_{SOUR} and R_{DUT} load. For example, if measuring a 100 kΩ on the 100 kΩ range, the **V**$_{MEAS}$ across the 100 kΩ will be 0.85 V (10 A to 1.5 µA) x 100 KΩ during $I_{OPENLEAD}$ and 1 V during measurement phase.

The tables below note timing with `dmm.opendetector = dmm.ON`.

Range	SHI and SLO IOPENLEAD Phase (ms)1	SHI Settle Time (ms)	Line Freq (Hz)	SHI Measurement Time (ms)	
1-10 kΩ	4.0	0.5	60	min	max
				0.0083	250
			50	0.010	240
100 kΩ	4.0	2.0	60	0.0083	250
			50	0.010	240
1 MΩ	4.0	30.0	60	0.0083	250
			50	0.010	240
10 MΩ to 100 MΩ	4.0	5.0[1]	60	0.0083	250

Range	Internal DMM Comm. (ms)	SLO Settle Time (ms)	SLO Measurement Time (ms)		Internal DMM Comm.
1 to 10 kΩ	0.06	0.5	min	max	0.06
			0.0083	250	
			0.010	240	
100 kΩ	0.06	1.0	0.0083	250	0.06
			0.010	240	
1 MΩ	0.06	1.0	0.0083	250	0.06
			0.010	240	

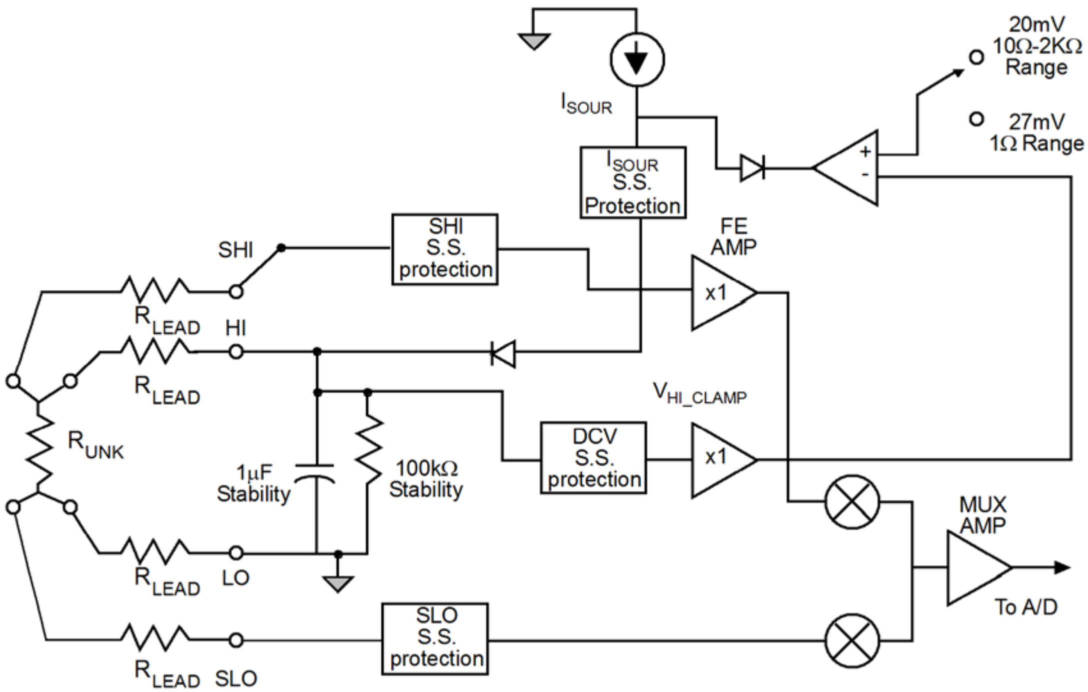
1. For 10 MΩ and 100 MΩ, $V_{measurement}$ is made on Input HI. Input Sense HI is unused.
2. Default condition for 4-wire is `dmm.opendetector=dmm.ON`.
3. For `dmm.drycircuit=dmm.ON`, $I_{OPENLEAD}$ is disabled, but `print(dmm.opendetector)` returns 1.0.
4. Additional cable and Series 3700A card capacitance can increase settling times, resulting in additional measurement uncertainty. Keithley Instruments recommend the use of Teflon or other low-dielectric absorption wire insulation for these measurements.

4-wire dry-circuit open lead detection

The Model 3706A dry-circuit resistance measurement circuitry was designed for low power, low glitch, and low open voltage applications such as GMRR head testing and air bag / squib testing that require low energy resistance sourcing. For this reason, `dmm.opendetector = dmm.ON` is disabled when `dmm.drycircuit = dmm.ON`. The $I_{OPENLEAD}$ current pulse would exceed the dry-circuit voltage application.

The follow schematic provides a simplified view of the Model 3706A 4-wire dry-circuit open lead detection.

Figure 107: Simplified Dry-Circuit open V-clamp feedback loop schematic



Dry-clamp open lead detector (dry-circuit)

A hardware detector is used to detect an open input lead. The hardware detector uses a internal circuit to monitor the voltage on the $V_{DRY\text{-}CLAMP}$ terminal. The circuit will stop the A/D in less than 100 µs and return an overflow reading if the voltage is greater than 1 V.

VMEAS open voltage (dry-circuit)

If either Input Sense HI or Sense LO V_{MEAS} is outside the enclosed table voltages, the A/D will stop in less than 100 µs and return an overflow reading.

Range	V_{MEAS} SHI or SLO High Limit Open Lead Detection	V_{MEAS} SHI or SLO Low Limit Open Lead Detection
1 Ω	> 27 mV	< –10 mV
10 Ω to 2 kΩ	> 20 mV	< –10 mV

Calculated measurement open voltage (dry-circuit)

A calculated measurement that exceeds 120% of the range will return an overflow reading.

> **NOTE**
>
> INPUT Sense HI is internally connected to INPUT HI. The connection allows proper open circuit voltage, even with Sense HI disconnected. With INPUT Sense HI disconnected, and the other inputs properly connected, the measurement will read the V_{DUT} and $R_{LEADVOLTAGE}$ drop.
>
> For `dmm.drycircuit = dmm.ON` and `dmm.opendetector = dmm.ON`, $I_{OPENLEAD}$ will be disabled, but a `print(dmm.opendetector)` will still return 1.0.

Open thermocouple detection

The Model 3706A open thermocouple detection works in similar fashion to the open lead detection. Refer to Open lead detection (on page 5-14). The open thermocouple detection performs as follows:

- V_{MEAS} open voltage: If Input HI V_{MEAS} is outside ±120 mV, the A/D will stop in less than 100 µs and return an overflow reading.
- A calculated measurement outside of the ranges in the following table will cause the "Overflow" message to be displayed.

Type	Range
J	-200 °C to +760 °C
K	-200 °C to +1372 °C
N	-200 °C to +1300 °C
T	-200 °C to +400 °C
E	-150 °C to +1000 °C
R	0 °C to +1768 °C
S	0 °C to +1786 °C
B	+350 °C to +1820 °C

- If during a measurement cycle, with dmm.opendetector = dmm.ON, the ohm's function [illegible]A I_{SOUR} is pulsed on and off before the start of each measurement. The A/D will monitor V_{MEAS} for 0.8 ms. During the IONPHASE, if a resistance of more than 1.15 kΩ is detected, or the input voltage is greater than 120 mV, the A/D will stop in less than 100 µs and return an overflow reading. If less than 1.15 kΩ is detected and the input voltage is in the range of ±120 mV, the open lead detection current is turned off and a normal thermocouple temperature measurement is performed (see Thermocouple connections (on page 4-26)).
- I_{SOUR} open voltage with `dmm.opendetector`. A hardware detector is used to continuously detect for open input lead. The hardware detector uses a comparator circuit to monitor the voltage on the ohm I_{SOUR} $V_{OPEN-HI-LEAD}$ terminal. If during a measurement cycle, the input voltage on I_{SOUR} $V_{OPEN -HI-LEAD}$ terminal is greater than 7.1 V, the A/D will stop in less than 100 µs and return an overflow reading. The following table notes timing with `dmm.opendetector = dmm.ON`.

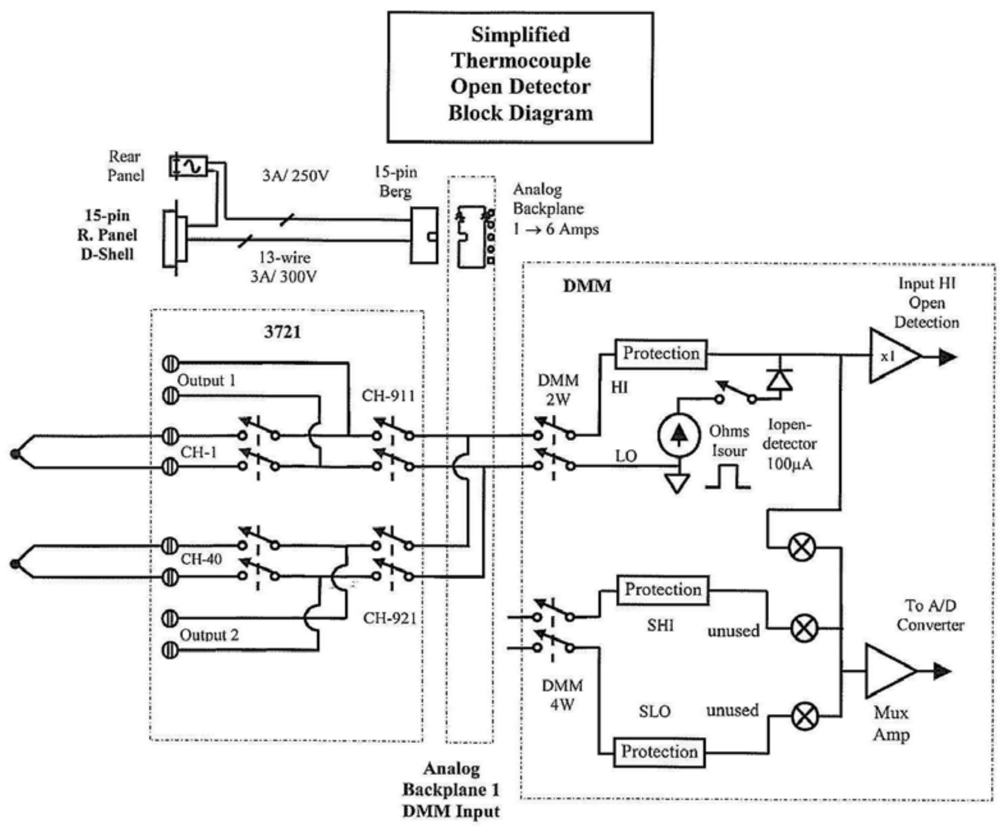
The thermocouple open detection times are listed in the following table.

Ion source settle (ms)	$I_{OPENLEAD}$ measure (ms)	Phase internal DMM comm. (ms)	I_{off} source settle (ms)	Line freq (Hz)	T/C measurement time (ms)		Internal DMM comm. (ms)
1.0	0.8	0.4	1.0	60	min	max	0.06
					0.0083	250	
				50	0.010	240	

1. Default condition is `dmm.opendetector=dmm.ON` or `1`.
2. For `dmm.transducer=dmm.TEMP_THERMISTOR`, `dmm.TEMP_THREERTD`, and `dmm.TEMP_FOURRTD`, $I_{OPENLEAD}$ phase is disabled, but `print(dmm.opendetector)` returns 1.0.
3. `dmm.opendetector` is a common remote command, shared with `fourwireohms`. To enable or disable `dmm.opendetector` for either function, the appropriate function must be selected before applying the new `dmm.opendetector` state. For example, to disable thermocouple open detection, send `dmm.func="temperature"` then `dmm.opendetector=0`.

The following figure is a schematic representation of the Model 3706A open thermocouple detection.

Figure 108: Thermocouple Open Detector drawing



Accuracy calculations

The following information discusses how to calculate accuracy for both DC and AC characteristics.

Calculating DC characteristics accuracy

DC characteristics accuracy is calculated as follows:

For >=1plc, Accuracy = ±(ppm of reading + ppm of range)
(ppm = parts per million and 10ppm = 0.001%)

As an example of how to calculate the actual reading limits, assume that you are measuring 5V on the 10V range. You can compute the reading limit range from one-year DCV accuracy specifications as follows:

Accuracy = ±(25ppm of reading + 2ppm of range)
±[(25ppm × 5V) + (2ppm × 10V)]
±(125µV + 20µV)
±145µV

Thus, the actual reading range is 5V+/-320uV or from 4.99968V to 5.00032V.Thus, the actual reading range is: 5V± 145µV or from 4.999855V to 5.000145V.

For <=1plc, Accuracy = +/-(ppm of reading + ppm of range + rms noise addr)

For example, to calculate the accuracy of the above example at 0.006plc:

Accuracy = +/- ((25ppm of reading) + (2ppm of range) + (2.5 x 7ppm of range))
= +/- ((25ppm x 5V) + (2ppm x 10V) + (2.5 x 7ppm x 10V))
= +/- (125uV + 20uV + 175uV)
= +/- 320uV

DC current and resistance calculations are performed in exactly the same manner using the pertinent specifications, ranges, and input signal values.

Calculating AC characteristics accuracy

AC characteristics accuracy is calculated similarly, except that AC specifications are given as follows:

Accuracy = (% of reading + % of range)

As an example of how to calculate the actual reading limits, assume that you are measuring 120V, 60Hz on the 300V range. You can compute the reading limit range from ACV one-year accuracy specifications as follows:

Accuracy = ±(0.06% of reading + 0.03% of range)
±[(0.0006 × 120V) + (0.0003 × 300V)]
±(0.072V + 0.09V)
±0.162V

In this case, the actual reading range is: 120V± 0.162V or from 119.838V to 120.162V.

AC current calculations are performed in exactly the same manner using the pertinent specifications, ranges, and input signal values.

Calculating dB characteristics accuracy

The relationship between voltage and dB is as follows:

$$dBm = 20_{log}\left(\frac{V_{in}}{V_{ref}}\right)$$

As an example of how to calculate the actual readings limits for dB, with a user-defined VREF of 10V, you must calculate the voltage accuracy and apply it to the above equation.

To calculate a -60dB measurement, assume 10mV RMS for a VREF of 10V. Using the 100mV range, one-year, 10Hz - 20kHz frequency band, and SLOW rate, the voltage limits are as follows:

Accuracy =	±[(0.06% of reading) + (0.03% of range)]
	±[(0.0006 × 10mV) + (0.0003 × 100mV)]
	±(6µV + 30µV)
	±36µV

Thus, the actual reading accuracy is 10mV ±36mV or 10.036mV to 9.964mV. Applying the voltage reading accuracy into the dB equation yields:

$$dBm = 20_{log}\left(\frac{10.036mV}{10V}\right) = -59.96879dB$$

$$dBm = 20_{log}\left(\frac{9.964mV}{10V}\right) = -60.03133dB$$

Thus, the actual reading accuracy is -60dB + 0.031213dB to -60dB - 0.031326dB.

dBm and dB for other voltage inputs can be calculated in exactly the same manner using pertinent specifications, ranges, and other reference voltages.

Additional derating factors

In some cases, additional derating factors must be applied to calculate certain accuracy values. For example, an additional derating for the following conditions:

1. -0.4mV with open inputs and the 10M-ohm divider enabled
2. +/-(8ppm or reading + 5uV) with autozero off for +/-1 degree C and <=10minutes
3. For 2-wire ohms, add 100m-ohm to "ppm of range" with REL
4. Add 0.1% to 10M-ohm range when measuring through a Series 3700A card >50% relative humidity

Before calculating accuracy, study the associated specifications very carefully to see if any derating factors apply.

Understanding Precision Time Protocol (PTP)

The Precision Time Protocol (PTP) is a time standard that does not have any discontinuities (that is, no leap seconds, time zones, or daylight savings). This is important for computing time deltas between events. Currently, the difference between PTP and UTC is 32 seconds.

The Model 3706A is not time-zone aware, just like your watch. For a stand-alone Model 3706A, PTP = UTC = local time. However, things can get confusing if the Model 3706A is synchronized to a device that is time-zone aware. The Model 3706A will still present UTC, but the other device will present UTC +/- offset as the local time, and so they will be different. As a result, you should always use PTP if possible. Programs using PTP will work correctly, regardless of the presence of time-zone aware devices.

The Model 3706A has two versions of time for most commands, `.seconds` and `.ptpseconds`, which represent UTC and PTP time. Use the `ptp.utcoffset` value to move between the two times.

NOTE

This value is zero unless the master clock populates it otherwise based on its information.

The following two statements produce the same value:

```
print(buffer.seconds[1] + ptp.utcoffset)
print(buffer.ptpseconds[1])
```

Example:

Run five scans once every hour starting at 3 a.m. tomorrow.

Assume tomorrow is Sept. 27, 2007. The first step is to convert the date and time to UTC format, and then to PTP.

```
 -- convert to UTC time
Start_time = os.time{year=2007, month=9, day=27, hour=3}
-- convert to PTP time
Start_time = Start_time + ptp.utcoffset
```

Set up the alarm as follows:

```
schedule.alarm[1].ptpseconds = Start_time
schedule.alarm[1].fractionalseconds = 0
schedule.alarm[1].repetition = 5
–– 1 hr = 60 sec.s x 60 mins
schedule.alarm[1].period = 60*60
schedule.alarm[1].enable = 1
```

Tie the above time event to a simple scan of DCV on channels 1 to 5 (in slot 1):

```
dmm.setconfig("1001:1005", "dcvolts")
scan.create("1001:1005")
-- 5 scans of 5 channels
buf = dmm.makebuffer(25)
-- initiates the scan start
scan.trigger.arm.stimulus = schedule.alarm[1].EVENT_ID
scan.scancount = 5
scan.background(buf)
```

The scan will initiate once the time condition is met.

Check the scan progress with the following command:

```
scan.state()
```

Section 6

Introduction to TSP operation

In this section:

Introduction to TSP operation

Instruments that are Test Script Processor (TSP®) enabled operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you would 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 . You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Controlling the instrument by sending individual command messages

The simplest method of controlling an instrument through the communication interface is to send it a message that contains remote commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants are commands that represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses ( ). You should only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Example 1

`beeper.beep(0.5, 2400)` `delay(0.250)` `beeper.beep(0.5, 2400)`	Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

You can use the results of a function-based command directly or assign variables to the results for later access. The following code defines `x` and prints it.

`x = math.abs(-100)` `print(x)`	Output: `100`

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, some characteristics of TSP-enabled instruments are the model number (`localnode.model`) and the state of the beeper (`beeper.enable`).

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 or a predefined constant.

Example 1: Set an attribute using a number

`beeper.enable = 0`	This attribute controls the beeps that occur when front-panel controls are selected. Setting this attribute to 0 turns off the beeper.

Example 2: Set an attribute using a constant

`format.data = format.REAL64`	Using the constant `REAL64` sets the print format to double precision floating point format.

To read an attribute, you can use the attribute as the parameter of a function, or assign it to another variable.

Example 3: Read an attribute using a function

`print(format.data)`	Reads the data format by passing the attribute to the print function. If the data format is set to 3, the output is: `3.00000e+00` This shows that the data format is set to double precision floating point.

Example 4: Read an attribute using a variable

`fd = format.data`	This reads the data format by assigning the attribute to a variable named `fd`.

Queries

Test Script Processor (TSP®) enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

`x = 10` `print(x)`	Example of an output response message: `1.00000e+01` Note that your output may be different if you set your ASCII precision setting to a different value.

Data retrieval commands

You can send data retrieval commands that return a comma-delimited string. For example, `channel.getcount(`*`channelList`*`)` returns a count for each item passed to it through its parameter, *`channelList`*.

The comma-delimited string that is returned starts with the lowest channel and goes to the highest channel on Slot 1. It then lists each subsequent slot until the highest slot is reached. After the channels are listed, the analog backplane relays are listed, starting with Bank 1 followed by each subsequent bank.

For example, assume there is a Model 3720 card installed in slot 4, returning 72 comma-delimited values. Send the following command:

```
print(channel.getclose("slot4"))
```

The first 60 values returned are the closed channel specifiers, starting with 1 and increasing to 60. The next six values are for analog backplane relays in Bank 1 (starting at 1 and increasing to 6). The final six values are for analog backplane relays in Bank 2 (starting at 1 and increasing to 6).

If the command was `channel.getstate()` instead of `channel.getclose()`, 72 zero (0) or one (1) values would be returned, with a 0 indicating that the channel or backplane is open, and a 1 indicating that it is closed. The first 60 values are for Channels 1 to 60 (starting at 1 and increasing to 60). The last 12 values are the backplane relays (starting with Bank 1, Relay 1, and increasing to Bank 2, Relay 6).

NOTE

If a channel is paired for 4-wire operation by its pole setting, the paired channel state is returned in parenthesis () after the primary channel. For example, if the card in Slot 4 is a Model 3720 and has the 4-pole attribute for all channels set, querying for the states of `"slot4"` returns 72 zero (0) and one (1) values, with the first 60 shown as the primary channel state (paired channel state); the 12 backplane relays follow.

Sample code and output:

```
channel.setpole("slot4", 4)
print(channel.getstate("slot4"))
```

Output from above code:

```
0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),
0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),0(0),
0,0,0,0,0,0,0,0,0,0,0,0
```

The Model 3721 card has three additional backplane relays for commonside ohms functionality. Use `"slotX"` or `"allslots"` to query settings on this card to return information for channels 1 to 40, 911 to 916, 921 to 926, and then 917, 927, and 928 in the response message (the three additional commonside ohms backplane relays are listed last).

For example, to print out the channel images on this card when it is in slot 2 after a reset, send the following:

```
reset()
print(channel.getimage('slot2'))
```

Output from above code:

```
2001;2002;2003;2004;2005;2006;2007;2008;2009;2010;2011;2012;2013;2014;2015;
2016;2017;2018;2019;2020;2021;2022;2023;2024;2025;2026;2027;2028;2029;2030;
2031;2032;2033;2034;2035;2036;2037;2038;2039;2040;2041;2042;2911;2912;2913;
2914;2915;2916;2921;2922;2923;2924;2925;2926;2917;2927;2928
```

NOTE

The commonside ohms backplane relays (2917, 2927, and 2928) are listed last (except for the Model 3721 card; for details see the Series 3700 Switch and Control Cards Reference Manual).

Information on scripting and programming

If you need information about using scripts with your TSP-enabled instrument, see Fundamentals of scripting for TSP (on page 7-1).

If you need information about using the Lua programming language with the instrument, see Fundamentals of programming for TSP (on page 7-14).

About TSP commands

This section contains an overview of the TSP commands for the instrument. The commands are organized into groups, with a brief description of each group. Each section contains links to the detailed descriptions for each command in the TSP command reference section of this documentation (see TSP commands (on page 8-10)).

Alarms

schedule.alarm[N].enable (on page 8-350)
schedule.alarm[N].EVENT_ID (on page 8-351)
schedule.alarm[N].fractionalseconds (on page 8-352)
schedule.alarm[N].period (on page 8-353)
schedule.alarm[N].ptpseconds (on page 8-353)
schedule.alarm[N].repetition (on page 8-354)
schedule.alarm[N].seconds (on page 8-355)
schedule.disable() (on page 8-355)

Bit manipulation and logic operations

The bit functions perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

Logic operations

The `bit.bitand()`, `bit.bitor()`, and `bit.bitxor()` functions in this group perform bitwise logic operations on two numbers. The Test Script Processor (TSP®) scripting engine performs the indicated logic operation on the binary equivalents of the two integers. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation is returned as an integer.

Bit operations

The rest of the functions in this group are used for operations on the bits of a given number. These functions can be used to:

- Clear a bit
- Toggle a bit
- Test a bit
- Set a bit or bit field
- Retrieve the weighted value of a bit or field value

All these functions use an index parameter to specify the bit position of the given number. The least significant bit of a given number has an index of 1, and the most significant bit has an index of 32.

NOTE

The Test Script Processor (TSP) scripting engine stores all numbers internally as IEEE Std 754 double-precision floating point values. The logical operations work on 32-bit integers. Any fractional bits are truncated. For numbers larger than 4294967295, only the lower 32 bits are used.

bit.bitand() (on page 8-11)
bit.bitor() (on page 8-11)
bit.bitxor() (on page 8-12)
bit.clear() (on page 8-13)
bit.get() (on page 8-13)
bit.getfield() (on page 8-14)
bit.set() (on page 8-15)
bit.setfield() (on page 8-15)
bit.test() (on page 8-16)
bit.toggle() (on page 8-17)

Channel

Channel functions and attributes allow you to adjust, select, open, and close channels. You can also set common channel attributes and set up channel patterns.

The channel functions and attributes are:

Data queue

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

The data queue in the Test Script Processor (TSP®) scripting engine is first-in, first-out (FIFO).

You can access data from the data queue even if a remote group or a node has overlapped operations in process.

Digital I/O

NOTE

The Models 2604A, 2614A, and 2634A do not have digital input/output lines. The commands to control the digital input/output lines are not available for these models.

The digital I/O port of the instrument can control external circuitry (such as a component handler for binning operations).

The I/O port has 14 lines. Each line can be at TTL logic state 1 (high) or 0 (low). See the pinout diagram in Digital I/O port (on page 2-28, on page 3-43) for additional information.

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

Display

DMM

Error queue

When errors and events occur, the error and status messages are placed in the error queue. Use the error queue commands to request error and status message information.

Event log

You can use the event log to view specific details about LAN triggering events.

File I/O

You can use the file I/O commands to open and close directories and files, write data, or to read a file on an installed USB flash drive. File I/O commands are organized into two groups:

- Commands that reside in the `fs` and `io` table, for example: `io.open()`, `io.close()`, `io.input()`, and `io.output()`. Use these commands to manage file system directories; open and close file descriptors; and perform basic I/O operations on a pair of default files (one input and one output).
- Commands that reside in the file descriptors (for example: *`fileVar`*`:seek()`, *`fileVar`*`:write()`, and *`fileVar`*`:read()`) operate exclusively on the file with which they are associated.

The root folder of the USB flash drive has the absolute path:

`"/usb1/"`

NOTE

You can use either the slash (`/`) or backslash (`\`) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you will generally need to use a double backslash (`\\`) when you are creating scripts or sending commands to the instrument.

For basic information about navigation and directory listing of files on a flash drive, see File system navigation (on page 6-12).

File descriptor commands for file I/O use a colon (:) to separate the command parts rather than a period (.), like the `io` commands.

File descriptors cannot be passed between nodes in a TSP-Link® system, so the `io.open()`, `fileVar::read()`, and `fileVar::write` commands are not accessible to the TSP-Link system. However, the default input and output files mentioned above allow for the execution of many file I/O operations without any reference to a file descriptor.

fileVar:close() (on page 8-252)
fileVar:flush() (on page 8-252)
fileVar:read() (on page 8-253)
fileVar:seek() (on page 8-254)
fileVar:write() (on page 8-254)
fs.chdir() (on page 8-258)
fs.cwd() (on page 8-258)
fs.is_dir() (on page 8-258)
fs.is_file() (on page 8-259)
fs.mkdir() (on page 8-259)
fs.readdir() (on page 8-260)
fs.rmdir() (on page 8-260)
io.close() (on page 8-262)
io.flush() (on page 8-262)
io.input() (on page 8-263)
io.open() (on page 8-264)
io.output() (on page 8-264)
io.read() (on page 8-265)
io.type() (on page 8-266)
io.write() (on page 8-266)
os.remove()
os.rename()

The following standard I/O commands are not supported at this time:

File	I/O
• *fileVar*:lines() • *fileVar*:setvbuf()	• io.lines() • io.popen()

File system navigation

The Model 3706A can use commands from the Lua `fs` library to navigate and list files that are available on a flash drive. These Lua commands are in the `fs` command group in the instrument.

The `fs` commands make the file system of any given node available to the entire TSP-Link® system. For example, you can use the command `node[5].fs.readdir(".")` to read the contents of the current working directory on node 5.

The root folder of the USB flash drive has the absolute path:

`"/usb1/"`

NOTE

You can use either the slash (/) or backslash (\) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you will generally need to use a double backslash (\\) when you are creating scripts or sending commands to the instrument.

The instrument supports the following Lua `fs` commands:

fs.chdir() (on page 8-258)
fs.cwd() (on page 8-258)
fs.is_dir() (on page 8-258)
fs.is_file() (on page 8-259)
fs.mkdir() (on page 8-259)
fs.readdir() (on page 8-260)
fs.rmdir() (on page 8-260)

The following Lua `fs` commands are not supported at this time:

```
fs.chmod()
fs.chown()
fs.stat()
```

GPIB

These commands store the GPIB address and indicate whether GPIB communication is enabled.

comm.gpib.enable (on page 8-110)
gpib.address (on page 8-261)

Instrument identification

These commands store strings that describe the instrument.

localnode.description (on page 8-293)
localnode.model (on page 8-295)
localnode.revision (on page 8-299)
localnode.serialno (on page 8-299)

LAN and LXI

The LAN commands have options that allow you to review and configure network settings.

The `lan.config.*` commands allow you to configure LAN settings over the remote interface. Note that you must send `lan.applysettings()` for the configuration settings to take effect.

The `lan.status.*` commands help you determine the status of the LAN.

The `lan.trigger[N].*` commands allow you to set up and assert trigger events that are sent over the LAN.

Other LAN commands allow you to reset the LAN, restore defaults, check LXI domain information, and enable or disable the Nagle algorithm.

Local node

Commands that allow you to set and read from the local node.

PTP

Use these functions and attributes to configure the IEEE Std 1588 Precision Time Protocol (PTP). IEEE-1588 allows multiple devices to synchronize time to a less than 10 ms accuracy. Further information on the protocol, operation, and terminology is available from the IEEE organization documentation and other third-party sources.

The Series 3700A commands support the 2008 IEEE-1588 standards, as indicated below.

Reading buffer

Reading buffers capture measurements, ranges, instrument status, and output states of the instrument.

bufferVar.appendmode (on page 8-18)
bufferVar.basetimefractional (on page 8-19)
bufferVar.basetimeseconds (on page 8-19)
bufferVar.cachemode (on page 8-20)
bufferVar.capacity (on page 8-21)
bufferVar.channels (on page 8-21)
bufferVar.clear() (on page 8-23)
bufferVar.clearcache() (on page 8-23)
bufferVar.collectchannels (on page 8-24)
bufferVar.collecttimestamps (on page 8-25)
bufferVar.dates (on page 8-26)
bufferVar.formattedreadings (on page 8-27)
bufferVar.fractionalseconds (on page 8-28)
bufferVar.n (on page 8-29)
bufferVar.ptpseconds (on page 8-30)
bufferVar.readings (on page 8-31)
bufferVar.relativetimestamps (on page 8-32)
bufferVar.seconds (on page 8-34)
bufferVar.statuses (on page 8-35)
bufferVar.times (on page 8-36)
bufferVar.timestampresolution (on page 8-37)
bufferVar.timestamps (on page 8-38)
bufferVar.units (on page 8-39)

Reset

Resets settings to their default settings.

digio.trigger[N].reset() (on page 8-126)
lan.reset() (on page 8-274)
localnode.reset() (on page 8-298)
reset() (on page 8-317)
timer.reset() (on page 8-421)
trigger.blender[N].reset() (on page 8-424)
trigger.timer[N].reset() (on page 8-433)
tsplink.trigger[N].reset() (on page 8-446)

Queries and response messages

You can use the `print()`, `printbuffer()`, and `printnumber()` functions to query the instrument and generate response messages. The format attributes control how the data is formatted for the print functions used.

The localnode commands determine if generated errors are automatically sent and if prompts are generated.

format.asciiprecision (on page 8-255)
format.byteorder (on page 8-256)
format.data (on page 8-257)
localnode.prompts (on page 8-296)
localnode.prompts4882 (on page 8-297)
localnode.showerrors (on page 8-300)
print() (on page 8-306)
printbuffer() (on page 8-307)
printnumber() (on page 8-310)

Saved setups

Use the saved setups commands to save and restore the configuration of the instrument. You can save or restore configurations to or from the instrument's nonvolatile memory or an installed USB flash drive. You can use the `setup.poweron` attribute to specify which setup is recalled when the instrument is turned on.

createconfigscript() (on page 8-115)
setup.cards() (on page 8-368)
setup.poweron (on page 8-369)
setup.recall() (on page 8-370)
setup.save() (on page 8-370)

Scan

The scan functions and attributes allow you to set up scanning over the remove interface.

scan.abort() (on page 8-318)
scan.add() (on page 8-319)
scan.addimagestep() (on page 8-321)
scan.addwrite() (on page 8-322)
scan.background() (on page 8-323)
scan.bypass (on page 8-324)
scan.create() (on page 8-325)
scan.execute() (on page 8-327)
scan.list() (on page 8-328)
scan.measurecount (on page 8-330)
scan.mode (on page 8-331)
scan.nobufferbackground() (on page 8-332)
scan.nobufferexecute() (on page 8-333)
scan.reset() (on page 8-334)
scan.scancount (on page 8-335)
scan.state() (on page 8-336)
scan.stepcount (on page 8-337)
scan.trigger.arm.clear() (on page 8-337)
scan.trigger.arm.set() (on page 8-338)
scan.trigger.arm.stimulus (on page 8-338)
scan.trigger.channel.clear() (on page 8-340)
scan.trigger.channel.set() (on page 8-341)
scan.trigger.channel.stimulus (on page 8-341)
scan.trigger.clear() (on page 8-343)
scan.trigger.measure.clear() (on page 8-344)
scan.trigger.measure.set() (on page 8-344)
scan.trigger.measure.stimulus (on page 8-345)
scan.trigger.sequence.clear() (on page 8-346)
scan.trigger.sequence.set() (on page 8-347)
scan.trigger.sequence.stimulus (on page 8-348)

Scripting

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument efficiently. These commands describe how to create, load, modify, run, and exit scripts.

For detail on using scripts, see Fundamentals of scripting for TSP (on page 7-1).

createconfigscript() (on page 8-115)
script.anonymous (on page 8-355)
script.delete() (on page 8-356)
script.load() (on page 8-357)
script.new() (on page 8-358)
script.newautorun() (on page 8-359)
script.restore() (on page 8-359)
script.run() (on page 8-360)
script.user.catalog() (on page 8-361)
scriptVar.autorun (on page 8-361)
scriptVar.list() (on page 8-362)
scriptVar.name (on page 8-363)
scriptVar.run() (on page 8-364)
scriptVar.save() (on page 8-365)
scriptVar.source (on page 8-365)

Status model

The status model is a set of status registers and queues. You can use the following commands to manipulate and monitor these registers and queues to view and control various instrument events.

status.condition (on page 8-391)
status.measurement.* (on page 8-393)
status.node_enable (on page 8-396)
status.node_event (on page 8-398)
status.operation.* (on page 8-399)
status.operation.user.* (on page 8-401)
status.questionable.* (on page 8-403)
status.request_enable (on page 8-405)
status.request_event (on page 8-407)
status.reset() (on page 8-409)
status.standard.* (on page 8-409)
status.system.* (on page 8-411)
status.system2.* (on page 8-413)
status.system3.* (on page 8-415)
status.system4.* (on page 8-417)
status.system5.* (on page 8-419)

Slot

The slot attributes configure and read the settings of the cards in the slots. You can also set up pseudocards.

slot[X].banks.matrix (on page 8-371)
slot[X].columns.matrix (on page 8-372)
slot[X].commonsideohms (on page 8-372)
slot[X].digio (on page 8-373)
slot[X].endchannel.* (on page 8-373)
slot[X].idn (on page 8-377)
slot[X].isolated (on page 8-380)
slot[X].matrix (on page 8-380)
slot[X].maxvoltage (on page 8-381)
slot[X].multiplexer (on page 8-381)
slot[X].poles.four (on page 8-382)
slot[X].poles.one (on page 8-383)
slot[X].poles.two (on page 8-384)
slot[X].pseudocard (on page 8-384)
slot[X].startchannel.* (on page 8-386)
slot[X].tempsensor (on page 8-390)
slot[X].thermal.state (on page 8-390)

Time

bufferVar.collecttimestamps (on page 8-25)
bufferVar.timestampresolution (on page 8-37)
delay() (on page 8-119)
gettimezone() (on page 8-261)
os.time()
settime() (on page 8-366)
settimezone() (on page 8-367)
timer.measure.t() (on page 8-421)
timer.reset() (on page 8-421)

Top level instrument controls

These commands work with other commands to control general instrument functions. They are also used to set and gather instrument information.

The beeper commands allow you to enable or disable and sound the instrument beeper.

`delay()` stops instrument operation for a specified period of time. It is typically used to soak a device at a specific voltage or current for a period of time.

`memory.available()` and `memory.used()` allow you to determine the amount of memory in the instrument.

The `os` commands are Lua functions that allow you to change directory and file names.

`opc()` sets the operation complete status bit when all overlapped commands are completed.

The `upgrade` functions allow you to upgrade or downgrade the Model 3706A firmware.

The `userstring` commands allow you to manage user-defined strings in nonvolatile memory.

`waitcomplete()` allows you to send a command to wait for all overlapped operations in a group to complete.

beeper.beep() (on page 8-10)
beeper.enable (on page 8-10)
delay() (on page 8-119)
memory.available() (on page 8-302)
memory.used() (on page 8-303)
opc() (on page 8-306)
upgrade.previous() (on page 8-462)
upgrade.unit() (on page 8-463)
userstring.add() (on page 8-463)
userstring.catalog() (on page 8-464)
userstring.delete() (on page 8-465)
userstring.get() (on page 8-465)
waitcomplete() (on page 8-466)

Triggering

The triggering commands allow you to set the conditions that the instrument uses to determine when measurements are captured.

digio.trigger[N].assert() (on page 8-121)
digio.trigger[N].clear() (on page 8-122)
digio.trigger[N].EVENT_ID (on page 8-122)
digio.trigger[N].mode (on page 8-123)
digio.trigger[N].overrun (on page 8-124)
digio.trigger[N].pulsewidth (on page 8-125)
digio.trigger[N].release() (on page 8-125)
digio.trigger[N].reset() (on page 8-126)
digio.trigger[N].stimulus (on page 8-127)
digio.trigger[N].wait() (on page 8-129)
display.trigger.EVENT_ID (on page 8-147)
lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].clear() (on page 8-282)
lan.trigger[N].connect() (on page 8-283)
lan.trigger[N].connected (on page 8-283)
lan.trigger[N].disconnect() (on page 8-284)
lan.trigger[N].EVENT_ID (on page 8-284)
lan.trigger[N].ipaddress (on page 8-285)
lan.trigger[N].mode (on page 8-286)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].protocol (on page 8-287)
lan.trigger[N].pseudostate (on page 8-288)
lan.trigger[N].stimulus (on page 8-288)
lan.trigger[N].wait() (on page 8-291)
trigger.blender[N].clear() (on page 8-422)
trigger.blender[N].EVENT_ID (on page 8-422)
trigger.blender[N].orenable (on page 8-423)
trigger.blender[N].overrun (on page 8-424)
trigger.blender[N].reset() (on page 8-424)
trigger.blender[N].stimulus[M] (on page 8-425)
trigger.blender[N].wait() (on page 8-427)
trigger.clear() (on page 8-428)
trigger.EVENT_ID (on page 8-428)
trigger.timer[N].clear() (on page 8-429)
trigger.timer[N].count (on page 8-429)
trigger.timer[N].delay (on page 8-430)
trigger.timer[N].delaylist (on page 8-430)
trigger.timer[N].EVENT_ID (on page 8-431)
trigger.timer[N].overrun (on page 8-431)
trigger.timer[N].passthrough (on page 8-432)
trigger.timer[N].reset() (on page 8-433)
trigger.timer[N].stimulus (on page 8-433)
trigger.timer[N].wait() (on page 8-435)
trigger.wait() (on page 8-435)
tsplink.trigger[N].assert() (on page 8-441)
tsplink.trigger[N].clear() (on page 8-441)
tsplink.trigger[N].EVENT_ID (on page 8-442)
tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].pulsewidth (on page 8-445)
tsplink.trigger[N].release() (on page 8-445)
tsplink.trigger[N].reset() (on page 8-446)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

TSP-Link

These functions and attributes allow you to set up and work with a system that is connected by a TSP-Link® network.

NOTE

The TSP-Link® is not available on the Models 2604A, 2614A, and 2634A. These commands are not available on those models.

TSP-Net

The TSP-Net module provides a simple socket-like programming interface to Test Script Processor (TSP®) enabled instruments.

Userstrings

Use the functions in this group to store and retrieve user-defined strings in nonvolatile memory. These strings are stored as key-value pairs. Key-value pairs are associated arrays of data items, where the key is used to index into the array. The key is a unique identifier such as a part number or identification string. The value is a data item or a pointer to where that data item is stored.

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.

userstring.add() (on page 8-463)
userstring.catalog() (on page 8-464)
userstring.delete() (on page 8-465)
userstring.get() (on page 8-465)

Section 7

Instrument programming

In this section:

Fundamentals of scripting for TSP

NOTE

Though it can improve your process to use scripts, you do not have to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages. The next few sections of the documentation describe scripting and programming features of the instrument. You only need to review this information if you are using scripting and programming.

Scripting helps you combine commands into a block of code that the instrument can run. Scripts help you communicate with the instrument more efficiently.

Scripts offer several advantages compared to sending individual commands from the host controller (computer):

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs more quickly and efficiently when it processes scripts than it does when it processes individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP®) scripting engine processes and runs scripts.

This section describes how to create, load, modify, and run scripts.

What is a script?

A script is a collection of instrument control commands and programming statements. Scripts that you create are referred to as **user scripts**.

Your scripts can be interactive. Interactive scripts display messages on the front panel of the instrument that prompt the operator to enter parameters.

Run-time and nonvolatile memory storage of scripts

Scripts are loaded into the run-time environment of the instrument. From there, they can be stored in the nonvolatile memory.

The run-time environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the run-time environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the run-time environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them even if the power is cycled. The scripts that are in nonvolatile memory are loaded into the run-time environment when the instrument is turned on.

Scripts are placed in the run-time environment when:

- The instrument is turned on. All scripts that are saved to nonvolatile memory are copied to the run-time environment when the instrument is turned on.
- Loaded over a remote command interface.

For detail on the amount of memory available in the run-time environment, see Memory considerations for the run-time environment (on page 7-43).

NOTE

If you make changes to a script in the run-time environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See Working with scripts in nonvolatile memory (on page 7-10).

What can be included in scripts?

Scripts can include combinations of TSP commands and Lua code. TSP commands instruct the instrument to do one thing and are described in the command reference (see TSP commands (on page 8-10)). Lua is a scripting language that is described in Fundamentals of programming for TSP (on page 7-14).

Commands that cannot be used in scripts

Though an instrument accepts the following commands, you cannot use these commands in scripts.

Commands that cannot be used in scripts

General commands	IEEE Std 488.2 common commands	
abort	*CLS	*RST
endflash	*ESE	*SRE
endscript	*ESE?	*SRE?
flash	*ESR?	*STB?
loadscript	*IDN?	*TRG
loadandrunscript	*OPC	*TST?
password	*OPC?	*WAI

Manage scripts

This section describes how to create scripts by sending commands over the remote interface and using TSB Embedded.

Tools for managing scripts

To manage scripts, you can send messages to the instrument, use your own development tool or program, use Keithley Instruments Test Script Builder (TSB) software, or use TSB Embedded on the instrument's web interface. TSB and TSB Embedded are described below.

- **Test Script Builder (TSB) software:** TSB software is a programming tool that is on the Test Script Builder Software Suite CD-ROM (included with your Series 3700A). You can use it to create, modify, debug, and store Test Script Processor (TSP®) scripting engine scripts. For more information about using the TSB software, see Using Test Script Builder (TSB) (on page 7-34).
- **TSB Embedded:** TSB Embedded is a tool with a reduced set of features than the complete Keithley TSB software. TSB Embedded has both script-building functionality and console functionality (single-line commands). It is accessed from a web browser.

NOTE

If you are using TSB or TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

Create and load a script

You create scripts by loading them into the run-time environment of the instrument. You can load a script as a named script or as the anonymous script.

Once a script is loaded into the instrument, you can execute it remotely or from the front panel.

Anonymous scripts

If a script is created with the `loadscript` or `loadandrunscript` command with no name defined, it is called the "anonymous" script. There can only be one anonymous script in the run-time environment. If another anonymous script is loaded into the run-time environment, it replaces the existing anonymous script.

Named scripts

A named script is a script with a unique name. You can have as many named scripts as needed in the instrument (within the limits of the memory available to the run-time environment). When a named script is loaded into the run-time environment with the `loadscript` or `loadandrunscript` commands, a global variable with the same name is created to reference the script.

Key points regarding named scripts:

- If you load a new script with the same name as an existing script, the existing script becomes an unnamed script, which in effect removes the existing script if there are no variables that reference it.
- Sending revised scripts with different names will not remove previously loaded scripts.
- Named scripts can be saved to internal nonvolatile memory. Saving a named script to nonvolatile memory allows the instrument to be turned off without losing the script. See Working with scripts in nonvolatile memory (on page 7-10).

Load a script by sending commands over the remote interface

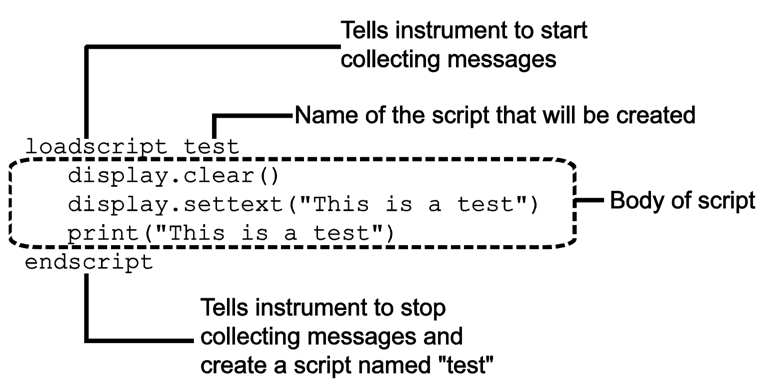
To load a script over the remote interface, you can use the `loadscript`, `loadandrunscript`, and `endscript` commands.

The `loadscript` and `loadandrunscript` commands start the collection of messages that make up the script. When the instrument receives either of these commands, it starts collecting all subsequent messages. Without these commands, the instrument would run them immediately as individual commands.

The `endscript` command tells the instrument to compile the collection of messages. It compiles the messages into one group of commands. This group of commands is loaded into the run-time environment.

The following figure shows an example of how to load a script named “test.” The first command tells the instrument to start collecting the messages for the script named “test.” The last command marks the end of the script. When this script is run, the message “This is a test” is displayed on the instrument and sent to the computer.

Figure 109: Loadscript and endscript example



To load a named 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. See Run scripts (on page 7-5).

NOTE

To run the script immediately, use `loadandrunscript` *`scriptName`* instead of `loadscript`.

Create a script using TSB Embedded

NOTE

If you are using TSB Embedded to create scripts, you do not need to use the commands `loadscript` or `loadandrunscript` and `endscript`.

You can create a script from the instrument web page with TSB Embedded. When you save the script in TSB Embedded, it is loaded into the run-time environment and saved in the nonvolatile memory of the instrument. For information about using TSB Embedded, select the **Help** button on a web page or the Help option from the navigation pane on the left side of the web interface.

To create a script using TSB Embedded:

1. In the TSP Script box, enter a name for the script.
2. In the input area, enter the sequence of commands to be included in the script.
3. Click **Save Script**. The name is added to the User Scripts list on the left.

Create a script using the create configuration script feature

The create configuration script feature captures the present settings of the instrument. Once saved, you can use this script to return to that configuration, or use it as a starting point to create your own scripts.

Once created, the configuration script is a normal TSP script — you can use it as you do any other script.

For detail on creating a configuration script, see Save the present configuration (on page 2-100).

Run scripts

This section describes how to run the anonymous and named scripts.

NOTE

If the instrument is in local control when the script is started, it switches to remote control (`REM` is displayed) while the script is running. The instrument is returned to local control when the script completes. If you press the front-panel **EXIT (LOCAL)** key while the script is running, the script is stopped.

Run the anonymous script

The anonymous script can be run many times without reloading it. It remains in the run-time environment until a new anonymous script is created or until the instrument is turned off.

To run the anonymous script, use any one of these commands:

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

Run a named script

You can run any named script that is in the run-time environment using one of the following commands:

- *`scriptVar`*`()`
- *`scriptVar`*`.run()`

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

To run a named script from TSB Embedded, select the script from the User Scripts list and click **Run**.

When a script is named, it can be accessed using the global variable *`scriptVar`*.

Example: Run a named script

`test3()`	If the script `test3` is loaded into the run-time environment, the instrument executes `test3`.

Scripts that run automatically

You can set up scripts to run automatically when you power on the instrument. To do this, either set the `autorun` attribute for the script to `yes` (see Autorun scripts (on page 7-6)), or create a script with the script name `autoexec` (see Autoexec script (on page 7-7)).

Autorun scripts

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun. The run order for autorun scripts is arbitrary, so make sure the run order is not important.

As shown in the example below, you can set a script to run automatically by setting the `.autorun` attribute of the script to `"yes"` and then saving the script.

Example:

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

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

To disable autorun, set the script's `.autorun` attribute to `"no"` and then save the script.

NOTE

The *scriptVar*.save() command saves the script to nonvolatile memory, which makes the change persistent through a power cycle. See Save a user script to nonvolatile memory (on page 7-10) for more detail.

Example: Set a script to run automatically

`test5.autorun = "yes"` `test5.save()`	Assume a script named `test5` is in the run-time environment. The next time the instrument is turned on, `test5` script automatically loads and runs.

Autoexec script

The autoexec script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts marked as autorun have run.

To create a script that executes automatically, create and load a new script and name it `autoexec`. See Create and load a script (on page 7-3).

NOTE

You must save the autoexec script to nonvolatile memory if you want to use it after instrument power has been turned off and then turned on again. See Save a user script to nonvolatile memory (on page 7-10) for more detail.

Example: Creating an autoexec script with loadscript command

`loadscript autoexec` `display.clear()` `display.settext("Hello from autoexec")` `endscript` `autoexec.save()`	Creates the script `autoexec`. Saves the `autoexec` script to nonvolatile memory. The next time the instrument is turned on, "`Hello from autoexec`" is displayed.

Example: Creating an autoexec script using TSB Embedded

`display.clear()` `display.settext("Hello from autoexec")`	In the TSP Script box, enter `autoexec`. Enter the code in the entry box. Click **Save Script**. Creates a new script that clears the display when the instrument is turned on and displays "`Hello from autoexec`."

Save the anonymous script as a named script

To save the anonymous script to nonvolatile memory, you must name it first.

To save the anonymous script as a named script:

1. To name the script, send the command `script.anonymous.name = "`*`myTest`*`"` (where *`myTest`* is the name of the script).
2. Send the `script.anonymous.save()` command to save *`myTest`* to nonvolatile memory.

Retrieve a user script

There are several ways to retrieve the source code of a user script:

- One line at a time: Use `scriptVar.list()` to retrieve the source code one line at a time
- Entire script: Use the `print(scriptVar.source)` command to retrieve the script source code as a single string
- Use TSB Embedded

See Create and load a script (on page 7-3) for information about recreating the script and loading it back into the instrument.

NOTE

To get a list of scripts that are in nonvolatile memory, use the script.user.catalog() (on page 8-361) function.

Retrieve source code one line at a time

To retrieve the source code one line at a time, send the `scriptVar.list()` command. When this command is received, the instrument sends the entire script. Each line of the script is sent as a separate response message. The output includes the `loadscript` or `loadandrunscript` and `endscript` keywords.

After retrieving the source code, you can modify and save the command lines as a user script under the same name or a new name.

To retrieve the source code of a script one line at a time, send the command:

```
scriptVar.list()
```

Where *scriptVar* is the name of the script.

NOTE

To retrieve the commands in the anonymous script, use `script.anonymous.list()`.

Example: Retrieve source code one line at a time

`test.list()`	Retrieve the source of a script named "`test`". The output will look similar to: `loadscript test` `display.clear()` `display.settext("This is a test")` `print("This is a test")` `endscript`

Retrieve a script as a single string

To retrieve the entire user script source code as a single string, use the *scriptVar*.source attribute. The `loadscript` or `loadandrunscript` and `endscript` keywords are not included.

To retrieve the source code as a single string, send the command:

```
print(scriptVar.source)
```

Where *scriptVar* is the name of the script.

Example: Retrieve the source code as a single string

`print(test.source)`	Retrieve the source of a script named "test". Output might look similar to: `display.clear()` `display.settext("This is a test") print("This is a test")`

Script example: Retrieve the content of scripts

This set of examples:

- Retrieves the source of a script using *scriptVar*.list()
- Retrieves the source of a script using *scriptVar*.source

Example: Retrieve the content of a script with scriptVar.list()

`test.list()`	Request a listing of the source of `test`. An example of the possible instrument output is shown here (note that the `loadscript` and `endscript` commands are included). Output: `loadscript scriptVarTest` `listTones = {100, 400, 800}` `for index in listTones do` `   beeper.beep(.5, listTones[index])` `end` `endscript`

Example: Retrieve the content of a script with scriptVar.source

`print(test.source)`	Request a listing of the source of the script named `test`. The `loadscript` and `endscript` commands are not included. An example of the possible instrument output is: `listTones = {100, 400, 800}` `for index in listTones do` `   beeper.beep(.5, listTones[index])` `end`

Working with scripts in nonvolatile memory

The Fundamentals of scripting for TSP (on page 7-1) section in this manual describes working with scripts, primarily in the run-time environment. You can also work with scripts in nonvolatile memory.

The run-time environment and nonvolatile memory are separate storage areas in the instrument. The information in the run-time environment is lost when the instrument is turned off. The nonvolatile memory remains intact when the instrument is turned off. When the instrument is turned on, information in nonvolatile memory is loaded into the run-time environment.

Save a user script

You can save scripts to nonvolatile memory using commands or TSB Embedded.

Only named scripts can be saved to nonvolatile memory. The anonymous script must be named before it can be saved to nonvolatile memory.

NOTE

If a script is not saved to nonvolatile memory, the script is lost when the instrument is turned off.

To save a script to nonvolatile memory:

1. Create and load a named script (see Create and load a script (on page 7-3)).
2. Do one of the following:
   - Send the command `scriptVar.save()`, where *`scriptVar`* is the name of the script.
   - In TSB Embedded, click **Save 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.

Delete user scripts

NOTE

These steps remove a script from nonvolatile memory. To completely remove a script from the instrument, there are additional steps you must take. See Delete user scripts from the instrument (on page 7-43).

To delete a script from nonvolatile memory using a remote interface:

You can delete the script from nonvolatile memory by sending either of the following commands:

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

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

To delete a script from nonvolatile memory using TSB Embedded:

1. In TSB Embedded, select the script from the User Scripts list.
2. Click **Delete**. There is no confirmation message.

Example: Delete a user script from nonvolatile memory

`script.delete("test8")`	Delete a user script named `test8` from nonvolatile memory.

Run a user script from the instrument front panel

From the instrument front panel, you can load and run a user script.

To run the code from the front panel and add it to the USER menu:

1. Press the **LOAD** key.
2. Select **USER**.
3. Select the user chunk from list and press the **ENTER** key. The chunk is loaded into the run-time environment.

NOTE

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

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

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

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

Figure 110: VARIABLE - NOTE

NOTE

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

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

Load a script from the instrument front panel

You can load scripts to the run-time environment of the instrument from a USB flash drive.

When you load a named script from the flash drive, the script is named using the name that follows the `loadscript` shell keyword (not the filename on the flash drive). The script is loaded into the `script.user.scripts` table.

If the loaded file does not contain `loadscript` and `endscript` shell keywords, the code is loaded as the anonymous script. Loading an unnamed script overwrites the existing anonymous script.

The file must be a valid script file. If not, an error message is posted and no further action is taken. You can view the errors on the front panel of the instrument.

To load a script from a USB flash drive:

1. Insert the flash drive into the USB port on the instrument.
2. Press the **MENU** key.
3. Select the **SCRIPT** option.
4. Select the **LOAD** option.
5. Select the **USB** option. A menu is displayed that lists the .tsp files and directories on the flash drive.
6. If the files are in a directory, use the navigation wheel to select the directory. A new menu is displayed that lists the .tsp files and directories in that directory.
7. Use the navigation wheel to select the .tsp file you want to load.
8. If the script has the same name as a script that is already in memory, you are prompted to overwrite the script.
   - Select "Yes" to continue.
   - Select "No" to return to the list of files. You must select a file to continue.

1. The SCRIPT ACTION menu is displayed.
2. Select **SAVE_INTERNAL**.
3. The **SAVE SCRIPT INTERNAL** prompt is displayed. Select **Yes** to save the file to nonvolatile memory. (This is the same as sending *`scriptVar`*`.save()` with no parameters.)
4. The SCRIPT ACTION menu is displayed again.
5. If you would like to set the script to run from the **RUN** button:
   a. Select **ACTIVE_FOR_RUN**. MAKE ACTIVE SCRIPT is displayed.
   b. Select **YES**.

1. Loading is complete. To return to the MAIN menu, press **EXIT (LOCAL)** until the MAIN menu is displayed.
2. If you selected **ACTIVE_FOR_RUN**, you can select **RUN** to run the script.

Save a script from the instrument front panel

You can save scripts from the run-time environment to nonvolatile memory from the instrument front panel.

NOTE

If you want to save the anonymous script to nonvolatile memory, you must name it first. See Save the anonymous script as a named script (on page 7-7).

To save a script to nonvolatile memory from the front panel:

1. Press the **MENU** key.
2. Select the **SCRIPT** option.
3. Select the **SAVE** option.

   A list of the scripts available to save is displayed. It may take a few seconds to display. The displayed list is from the `script.user.scripts` table in the instrument.
4. Turn the navigation wheel to select the script that you want to save.
5. Select **INTERNAL**. Press the navigation wheel. The script is saved to nonvolatile memory using the script's name attribute.
6. Press **EXIT (LOCAL)** several times to return to the Main Menu.

Interactive script

An interactive script prompts the operator to input values using the instrument front panel. The following example script uses display messages to prompt the operator to:

- Enter the digital I/O line on which to output a trigger
- Enter the output trigger pulsewidth

After the output trigger occurs, the front display displays a message to the operator.

When an input prompt is displayed, the script waits until the operator inputs the parameter or presses the **ENTER** key.

The example shown here assumes that you are using TSB or TSB Embedded. If you are using a remote interface, you need to add the `loadscript` and `endscript` commands to the example code. See Load a script by sending commands over the remote interface (on page 7-4) for details.

```
-- Clear the display.
display.clear()

-- Prompt user for digital I/O line on which to output trigger.
myDigioLine = display.menu("Select digio line", "1 2 3 4 5 6 7 8 9")

-- Convert user input to a number.
intMyDigioLine = tonumber(myDigioLine)

-- Prompt user for digital output trigger mode.
myDigioEdge = display.menu("Select digio mode", "Rising Falling")
if myDigioEdge == "Rising" then
   edgeMode = digio.TRIG_RISING
else
   edgeMode = digio.TRIG_FALLING
end

-- Prompt user for output trigger pulsewidth.
myPulseWidth = display.prompt(
   "000.0", "us", "Enter trigger pulsewidth", 10, 10, 100)

-- Scale the entered pulsewidth.
myPulseWidth = myPulseWidth * 1e-6

-- Generate the pulse.
digio.trigger[intMyDigioLine].mode = edgeMode
digio.trigger[intMyDigioLine].pulsewidth = myPulseWidth
digio.trigger[intMyDigioLine].assert()

-- Alert the user through the display that the
-- output trigger has occurred.
display.setcursor(1, 1)
display.settext("Trigger asserted $Non digital I/O line " .. intMyDigioLine)

-- Wait five seconds and then return to main screen.
delay(5)
display.screen = display.MAIN
```

Fundamentals of programming for TSP

Introduction

To conduct a test, a computer (controller) is programmed to send sequences of commands to an instrument. The controller orchestrates the actions of the instrumentation. The controller is typically programmed to request measurement results from the instrumentation and make test sequence decisions based on those measurements.

To take advantage of the advanced features of the instrument, you can add programming commands to your scripts. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

The Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands.

What is Lua?

Lua is a programming language that can be used with TSP-enabled instruments. Lua is an efficient language with simple syntax that is easy to learn.

Lua is also a scripting language, which means that scripts are compiled and run when they are sent to the instrument. You do not compile them before sending them to the instrument.

Lua basics

This section contains the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts quickly.

For more information about Lua, see the Lua website (*http://www.lua.org*). Another source of useful information is the Lua users group (*http://lua-users.org*), created for and by users of Lua programming language.

Comments

Comments start anywhere outside a string with a double hyphen (--). If the text immediately after a double hyphen (--) is anything other than double left brackets (`[[`), the comment is a short comment, which continues only until the end of the line. If double left brackets (`[[`) follow the double hyphen (--), it is a long comment, which continues until the corresponding double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ ... ]]` pairs. The table below shows how to use code comments.

Using code comments

Type of comment	Comment delimiters	Usage	Example
Short comment	`--`	Use when the comment text is short enough that it will not wrap to a second line.	`--Disable the beeper.` `beeper.enable = beeper.OFF`
Long comment	`--[[ ]]`	Use when the comment text is long enough that it wraps to additional lines.	`--[[Display a menu with three menu items. If the second menu item is selected, the selection will be given the value Test2.]]`

Function and variable name restrictions

You cannot use Lua reserved words and top level command names for function or variable names.

You cannot use the following Lua reserved words for function or variable names.

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

You also cannot use top-level command names as variable names. If you use these names, it will result in the loss of use of the commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you turn the instrument power off and then turn it on again. These groups include:

Top level command names	
beeper	lan
bit	localnode
channel	opc
dataqueue	reset
delay	scan
digio	slot
display	status
errorqueue	timer
eventlog	trigger
exit	tsplink
format	tspnet
fs	userstring
gpib	waitcomplete
io	

Values and variable types

In Lua, you use variables to store values in the run-time environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Variable types

Variables can be one of the following types.

Variable types and values

Variable type returned	Value	Notes
`"nil"`	not declared	The type of the value `nil`, whose main property is to be different from any other value; usually it represents the absence of a useful value.
`"boolean"`	true or false	Boolean is the type of the values `false` and `true`. In Lua, both `nil` and `false` make a condition `false`; any other value makes it `true`.
`"number"`	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
`"string"`	sequence of words or characters	
`"function"`	a block of code	Functions perform a task or compute and return values.
`"table"`	an array	New tables are created with { } braces. For example, {1, 2, 3.00e0}.

To determine the type of a variable, you can call the `type()` function, as shown in the examples below.

NOTE

The output you get from these examples may vary depending on the data format that is set.

Example: Nil

`x = nil` `print(x, type(x))`	`nil nil`

Example: Boolean

`y = false` `print(y, type(y))`	`false boolean`

Example: 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. `1.30 number`

Example: Function

`function add_two(first_value,` `   second_value)` `   return first_value + second_value` `end` `print(add_two(3, 4), type(add_two))`	`7     function`

Example: Table

`atable = {1, 2, 3, 4}` `print(atable, type(atable))` `print(atable[1])` `print(atable[4])`	Defines a table with four numeric elements. Note that the "table" value (shown here as `a096cd30`) will vary. `table: a096cd30      table` `1` `4`

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the run-time environment.

Functions

With Lua, you can group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

Note that when a function is defined, it is stored in the run-time environment. Like all data that is stored in the run-time environment, the function persists until it is removed from the run-time environment, is overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody* is the code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE

The output you get from these examples will vary depending on the data format settings of the instrument.

Example 1

`function add_two(first_value,` `  second_value)` `  return first_value + second_value` `end` `print(add_two(3, 4))`	Creates a variable named `add_two` that has a variable type of function. Output: `7`

Example 2

`add_three = function(first_value,` `  second_value, third_value)` `  return first_value + second_value +` `    third_value` `end` `print(add_three(3, 4, 5))`	Creates a variable named `add_three` that has a variable type of function. Output: `12`

Example 3

`function sum_diff_ratio(first_value,` `  second_value)` `  psum = first_value + second_value` `  pdif = first_value - second_value` `  prat = first_value / second_value` `  return psum, pdif, prat` `end` `sum, diff, ratio = sum_diff_ratio(2, 3)` `print(sum)` `print(diff)` `print(ratio)`	Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it). Output: `5` `-1` `0.66666666666667`

Create functions using scripts

You can use scripts to define functions. Scripts that define a function are like any other script: They do not cause any action to be performed on the instrument until they are executed. The global variable of the function does not exist until the script that created the function is executed.

A script can consist of one or more functions. Once a script has been run, the computer can call functions that are in the script directly.

NOTE

The following steps use TSB Embedded. You can also use the `loadscript` and `endscript` commands to create the script over the remote interface. See Load a script by sending commands over the remote interface (on page 7-4).

Steps to create a function using a script:

1. In TSB Embedded, enter a name into the TSP Script box. For example, type `MakeMyFunction`.
2. Enter the function as the body of the script. This example concatenates two strings:
```
MyFunction = function (who)
   print("Hello " .. who)
end
```
3. Click **Save Script**.

   `MakeMyFunction` is now on the instrument in a global variable with the same name as the script (`MakeMyFunction`). However, the function defined in the script does not yet exist because the script has not been executed.
4. Run the script as a function. For this example, send:
```
MakeMyFunction()
```
   This instructs the instrument to run the script, which creates the `MyFunction` global variable. This variable is of the type `"function"` (see Variable types (on page 7-17)).
5. Run the new function with a value.
```
MyFunction("world")
```
   The response message is:
```
Hello world
```

Group commands using the function keyword

The following script contains instrument commands that display the name of the person that is using the script on the front panel of the instrument. It takes one parameter to represent this name. When this script is run, the function is loaded in memory. Once loaded into memory, you can call the function outside of the script to execute it.

When calling the function, you must specify a string for the *name* argument of the function. For example, to set the name to **John**, call the function as follows:

```
myDisplay("John")
```

Example: User script

User script created in Test Script Builder or TSB Embedded	User script created in user's own program
`function myDisplay(name)` `   display.clear()` `   display.settext(` `      name .. "$N is here!")` `end`	`loadscript` `function myDisplay(name)` `   display.clear()` `   display.settext(` `      name .. " $N is here!")` `end` `endscript`

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
–	subtraction
*	multiplication
/	division
-	negation (for example, c = –a)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~=	not equal
==	equal

Logical 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

`print(10 or errorqueue.next())`	`1.00000e+01`
`print(nil or "a")`	`a`
`print(nil and 10)`	`nil`
`print(false and errorqueue.next())`	`false`
`print(false and nil)`	`false`
`print(false or nil)`	`nil`
`print(10 and 20)`	`2.00000e+01`

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")`	`23` `Hello World`

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

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

You can use parentheses to change the precedences in an expression. The concatenation ("`..`") and exponentiation ("^") operators are right associative. All other binary operators are left associative. The examples below show equivalent expressions.

Equivalent expressions

`reading + offset < testValue/2+0.5`	=	`(reading + offset) < ((testValue/2)+0.5)`
`3+reading^2*4`	=	`3+((reading^2)*4)`
`Rdg < maxRdg and lastRdg <= expectedRdg`	=	`(Rdg < maxRdg) and (lastRdg <= expectedRdg)`
`-reading^2`	=	`-(reading^2)`
`reading^testAdjustment^2`	=	`reading^(testAdjustment^2)`

Conditional branching

Lua uses the `if`, `else`, `elseif`, `then`, and `end` keywords to do conditional branching.

Note that in Lua, `nil` and `false` are `false` and everything else is `true`. Zero (0) is `true` in Lua.

The syntax of a conditional block is as follows:

```
if expression then
   block
elseif expression then
   block
else
   block
end
```

Where:

- `expression` is Lua code that evaluates to either `true` or `false`
- `block` consists of one or more Lua statements

Example: If

`if 0 then` `   print("Zero is true!")` `else` `   print("Zero is false.")` `end`	Output: `Zero is true!`

Example: Comparison

`x = 1` `y = 2` `if x and y then` `   print("Both x and y are true")` `end`	Output: `Both x and y are true`

Example: If and else

`x = 2` `if not x then` `   print("This is from the if block")` `else` `   print("This is from the else block")` `end`	Output: `This is from the else block`

Example: Else and elseif

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

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

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

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

Output:

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

Loop control

If you need to repeat code execution, you can use the Lua `while`, `repeat`, and `for` control structures. To exit a loop, you can use the `break` keyword.

While loops

To use conditional expressions to determine whether to execute or end a loop, you use `while` loops. These loops are similar to Conditional branching (on page 7-22) statements.

```
while expression do
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: While

`list = {` `   "One", "Two", "Three", "Four", "Five", "Six"}` `print("Count list elements on numeric index:")` `element = 1` `while list[element] do` `   print(element, list[element])` `   element = element + 1` `end`	This loop exits when `list[element] = nil`. Output: `Count list elements on` `   numeric index:` `1   One` `2   Two` `3   Three` `4   Four` `5   Five` `6   Six`

Repeat until loops

To repeat a command, you use the `repeat` ... `until` statement. The body of a `repeat` statement always executes at least once. It stops repeating when the conditions of the `until` clause are met.

```
repeat
   block
until expression
```

Where:

- *`block`* consists of one or more Lua statements
- *`expression`* is Lua code that evaluates to either `true` or `false`

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Repeat until

`list = {` `   "One", "Two", "Three", "Four", "Five", "Six"}` `print("Count elements in list using repeat:")` `element = 1` `repeat` `   print(element, list[element])` `   element = element + 1` `until not list[element]`	Output: `Count elements in list` `   using repeat:` `1   One` `2   Two` `3   Three` `4   Four` `5   Five` `6   Six`

For loops

There are two variations of `for` statements supported in Lua: numeric and generic.

NOTE

In a `for` loop, the loop expressions are evaluated once, before the loop starts.

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Numeric for

```
list = {"One", "Two", "Three", "Four", "Five", "Six"}
---------- For loop -----------
print("Counting from one to three:")
for element = 1, 3 do
   print(element, list[element])
end
print("Counting from one to four, in steps of two:")
for element = 1, 4, 2 do
   print(element, list[element])
end
```

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

Output:

```
Counting from one to three:
1   One
2   Two
3   Three
Counting from one to four, in steps of two:
1   One
3   Three
```

Example: Generic for

```
days = {"Sunday",
   "Monday",     "Tuesday",
   "Wednesday",  "Thursday",
   "Friday",     "Saturday"}

for i, v in ipairs(days) do
   print(days[i], i, v)
end
```

The generic `for` statement works by using functions called iterators. On each iteration, the iterator function is called to produce a new value, stopping when this new value is nil.

Output:

```
Sunday     1    Sunday
Monday     2    Monday
Tuesday    3    Tuesday
Wednesday  4    Wednesday
Thursday   5    Thursday
Friday     6    Friday
Saturday   7    Saturday
```

Break

The `break` statement can be used to terminate the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is necessary to return or break in the middle of a block, an explicit inner block can be used.

NOTE

The output you get from these examples may vary depending on the data format settings of the instrument.

Example: Break with while statement

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a `break` value (`breakValue`) so that the `break` statement is used to exit the `while` loop before the value of k reaches `0`.

Output:

```
Going to break and k = 3
```

Example: Break with while statement enclosed by comment delimiters

```
local numTable = {5, 4, 3, 2, 1}
local k = table.getn(numTable)
--local breakValue = 3
while k > 0 do
   if numTable[k] == breakValue then
      print("Going to break and k = ", k)
      break
   end
   k = k - 1
end
if k == 0 then
   print("Break value not found")
end
```

This example defines a break value (`breakValue`), but the break value line is preceded by comment delimiters so that the break value is not assigned, and the code reaches the value `0` to exit the `while` loop.

Output:

```
Break value not found
```

Example: Break with infinite loop

`a, b = 0, 1` `while true do` `   print(a, b)` `   a, b = b, a + b` `   if a > 500 then` `      break` `   end` `end`	This example uses a `break` statement that causes the while loop to exit if the value of a becomes greater than 500. Output: `0     1` `1     1` `1     2` `2     3` `3     5` `5     8` `8     13` `13    21` `21    34` `34    55` `55    89` `89    144` `144   233` `233   377` `377   610`

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indices start with 1. Tables can be indexed not only with numbers, but with any value except `nil`. Tables can be heterogeneous, which means that they can contain values of all types except `nil`.

Tables are the sole data structuring mechanism in Lua. They may be used to represent ordinary arrays, symbol tables, sets, records, graphs, trees, and so on. To represent records, Lua uses the field `name` as an index. The language supports this representation by providing `a.name` as an easier way to express `a["name"]`.

NOTE

The output you get from this example may vary depending on the data format settings of the instrument.

Example: Loop array

`atable = {1, 2, 3, 4}` `i = 1` `while atable[i] do` `   print(atable[i])` `   i = i + 1` `end`	Defines a table with four numeric elements. Loops through the array and prints each element. The Boolean value of `atable[index]` evaluates to `true` if there is an element at that index. If there is no element at that index, `nil` is returned (`nil` is considered to be `false`). **Output:** 1 2 3 4

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP®) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

The following topics provide information on some of the basic Lua standard libraries. For additional information, see the Lua website (*http://www.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
`collectgarbage()` `collectgarbage(`*`limit`*`)`	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to zero (0), which forces a garbage-collection cycle. See the "Lua memory management" topic below for more information.
`gcinfo()`	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP®) scripting engine is using, and returns the present garbage collector threshold (also in kilobytes). See the "Lua memory management" topic below for more information.
`tonumber(`*`x`*`)` `tonumber(`*`x, base`*`)`	Returns *x* converted to a number. If *x* is already a number, or a convertible string, the number is returned; otherwise, it returns `nil`. An optional argument specifies the base to use when interpreting the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter A (in either upper or lower case) represents 10, B represents 11, and so forth, with Z representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(`*`x`*`)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(`*`v`*`)`	Returns (as a string) the type of its only argument. The possible results of this function are `"nil"` (a string, not the value `nil`), `"number"`, `"string"`, `"boolean"`, `"table"`, `"function"`, `"thread"`, and `"userdata"`.

Lua memory management

Lua automatically manages memory, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua occasionally runs a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted and the threshold is reset to twice the new value of the byte counter.

String library functions

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

String library functions

Function	Description
`string.byte(`*s*`)` `string.byte(`*s*`, `*i*`)` `string.byte(`*s*`, `*i*`, `*j*`)`	Returns the internal numeric codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for *i* is `1`; the default value for *j* is *i*.
`string.char(···)`	Receives zero or more integers. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.
`string.format(` *`formatstring`*`,` `...)`	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, n, p, and h are not supported and there is an extra option, q. The q option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written. For example, the call: `string.format('%q', 'a string with "quotes" and \n new line')` will produce the string: `"a string with \"quotes\" and \ new line"` The options `c`, d, `E`, e, `f`, g, `G`, `i`, o, u, `X`, and `x` all expect a number as argument. q and s expect a string. This function does not accept string values containing embedded zeros, except as arguments to the q option.
`string.len(`*s*`)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.

String library functions

Function	Description
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
`string.rep(s, n)`	Returns a string that is the concatenation of *n* copies of the string *s*.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of *s* that starts at *i* and continues until *j*; *i* and *j* can be negative. If *j* is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of *s* with length *j*, and `string.sub(s, -i)` returns a suffix of *s* with length *i*.
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.
`math.atan2(y, x)`	Returns the arc tangent of `y/x`, but uses the signs of both parameters to find the quadrant of the result (it also handles correctly the case of *x* being zero).
`math.ceil(x)`	Returns the smallest integer larger than or equal to *x*.
`math.cos(x)`	Returns the cosine of *x*.
`math.deg(x)`	Returns the angle *x* (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to *x*.
`math.frexp(x)`	Returns `m` and `e` such that $x = m2^e$, where `e` is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when *x* is zero).
`math.ldexp(x, n)`	Returns $m2^e$ (e should be an integer).
`math.log(x)`	Returns the natural logarithm of *x*.
`math.log10(x)`	Returns the base-10 logarithm of *x*.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle *x* (given in degrees) in radians.
`math.random()` `math.random(m)` `math.random(m, n)`	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C. When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer number *m*, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integer numbers *m* and *n*, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.
`math.randomseed(x)`	Sets *x* as the seed for the pseudorandom generator: equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the sine of *x*.
`math.sqrt(x)`	Returns the square root of *x*. (You can also use the expression `x^0.5` to compute this value.)
`math.tan(x)`	Returns the tangent of *x*.

Programming example

Programming example: Script with a for loop

The following script puts a message on the front panel display slowly — one character at a time. The intent of this example is to demonstrate:

- The use of a `for` loop
- Simple display remote commands
- Simple Lua string manipulation

NOTE

When creating a script using the TSB Embedded, you do not need the shell commands `loadscript` and `endscript`, as shown in the examples below.

Example: User script

User script created in TSB Embedded	User script created in user's own program
  `display.clear()` `myMessage = "Hello World!"` `for k = 1, string.len(myMessage) do` `   x = string.sub(myMessage, k, k)` `   display.settext(x)` `   print(x)` `   delay(1)` `end`	`loadscript` `display.clear()` `myMessage = "Hello World!"` `for k = 1, string.len(myMessage) do` `   x = string.sub(myMessage, k, k)` `   display.settext(x)` `   print(x)` `   delay(1)` `end` `endscript`

Using Test Script Builder (TSB)

Keithley Instruments Test Script Builder (TSB) is a software tool that simplifies building test scripts. You can use TSB to perform the following operations:

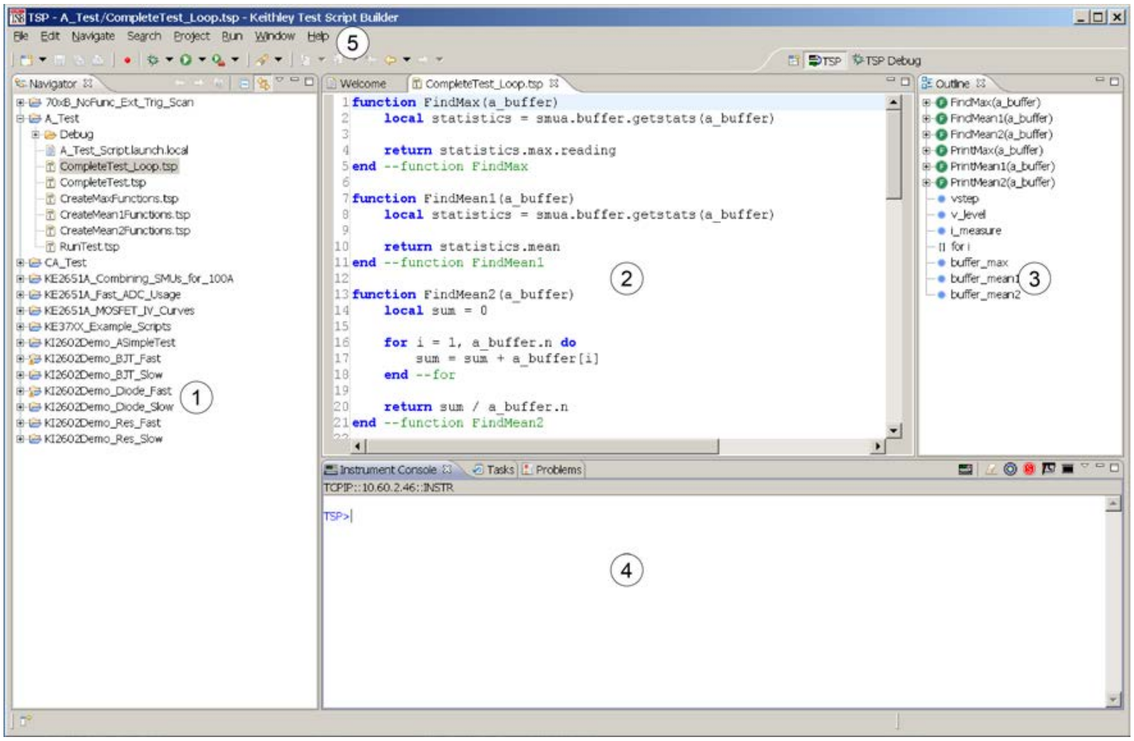
- Send remote commands and Lua statements
- Receive responses (data) from commands and scripts
- Upgrade instrument firmware
- Create, manage, and run user scripts
- Debug scripts
- Import factory scripts to view or edit and convert to user scripts

The Keithley Instruments Test Script Processor (TSP®) scripting engine is a Lua interpreter. In TSP-enabled instruments, the Lua programming language has been extended with Keithley-specific instrument control commands. For more information about using the Lua scripting language with Keithley TSP-enabled instruments, refer to the Fundamentals of programming for TSP (on page 7-14) section.

Keithley has created a collection of remote commands specifically for use with Keithley TSP-enabled instruments; for detailed information about those commands, refer to the "Command reference" section of the documentation for your specific instrument. You can build scripts from a combination of these commands and Lua programming statements. Scripts that you create are referred to as "user scripts." Also, some TSP-enabled instruments come with a number of built-in factory scripts.The following figure shows an example of the Test Script Builder. As shown, the workspace is divided into these areas:

- Project navigator
- Script editor
- Outline view
- Programming interaction
- Help files

Figure 111: 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

Installing the TSB software

The installation files for the Test Script Builder software are available on the Test Script Builder Software Suite CD (Keithley Instruments part number KTS-850 F01 or later) that came with your Model 3706A. You can also get it from the Keithley Instruments support website (*http://www.keithley.com/support*).

To install the Test Script Builder (TSB) software:

1. Close all programs.
2. Place the Test Script Builder Software Suite CD into your CD-ROM drive or start the software from the downloaded file.
3. Follow the on-screen instructions.

If you are using the CD-ROM and the web browser does not start automatically and display a screen with software installation links, open the installation file (`setup.exe`) located on the CD-ROM to start installation.

Project navigator

The project navigator consists of project folders and the script files (.tsp) created for each project. Each project folder can have one or more script files.

To view the script files in a project folder, click the plus (+) next to the project folder. To hide the folder contents, click the minus (–) next to the project folder.

You can download a TSP project to the instrument and run it, or you can run it from the TSB interface.

Script editor

The script editor is where scripts are written, modified, and debugged.

To open and display a script file, double-click the file name in the project navigator. You can have multiple script files open in the script editor at the same time. Each open script file is displayed on a separate tab.

To display another script file that is already open, click the tab that contains the script in the script editor area.

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.

Advanced scripting for TSP

The following topics describe advanced information that can help you understand how the Test Script Processor (TSP®) scripting engine works.

Global variables and the script.user.scripts table

When working with script commands, it is helpful to understand how scripts are handled in the instrument.

Scripts are loaded into the run-time environment from nonvolatile memory when you turn the instrument on. They are also added to the run-time environment when you load them into the instrument.

A script in the run-time environment can be:

- A named script
- An unnamed script
- The anonymous script (which is a special unnamed script)

Script names can be assigned by using the loadscript command or by defining the scriptVar parameter of the `script.new()` function. When a named script is loaded into the run-time environment:

- A global variable with the same name is created so that you can reference the script more conveniently.
- An entry for the script is added to the `script.user.scripts` table.

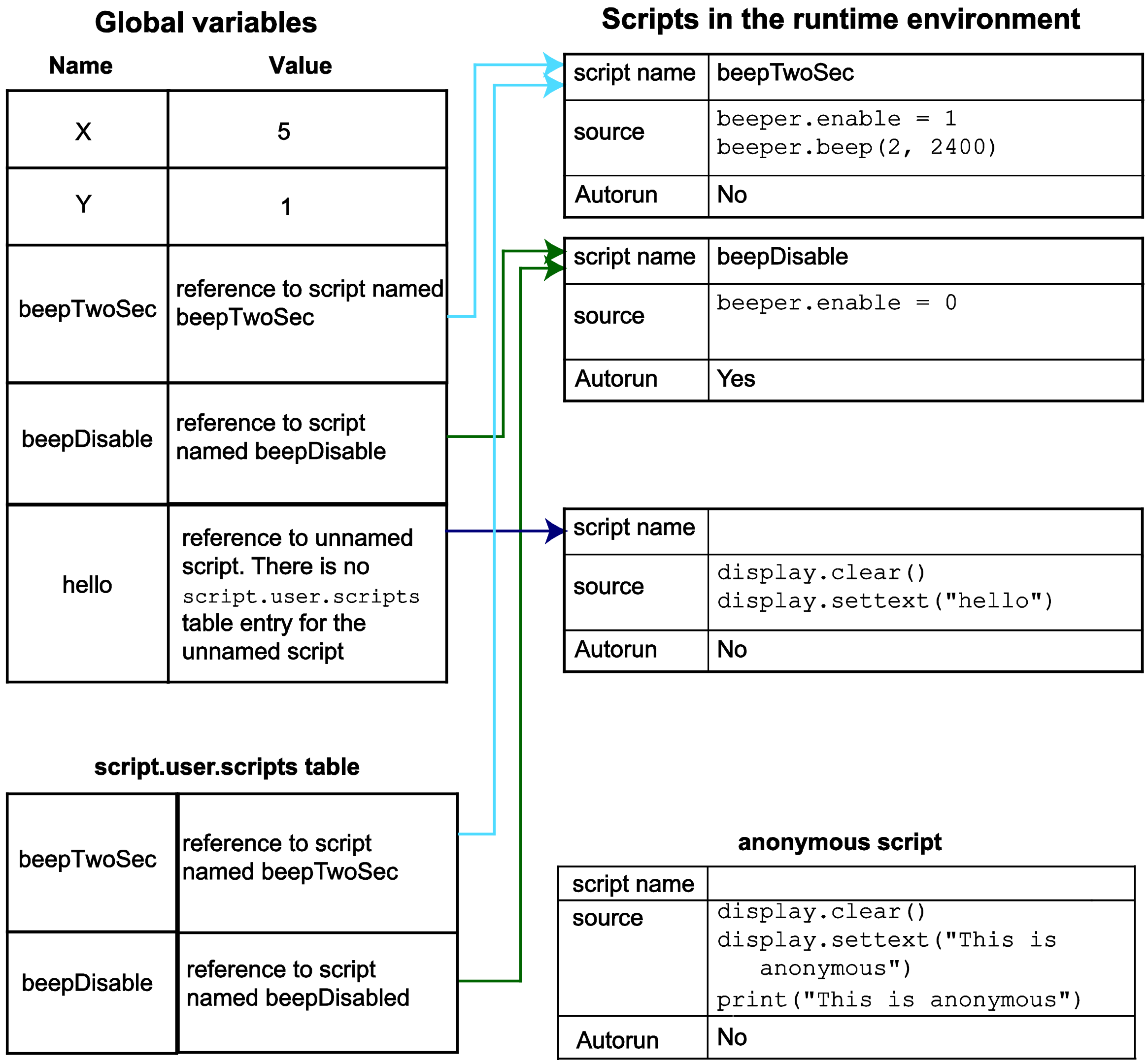
When you create a script using the `script.new()` function without providing a name, the script is added to the run-time environment as an unnamed script. The `script.new()` function returns the script, but the script is not added to the `script.user.scripts` table.

When the anonymous script is loaded, it does not have a global variable or an entry in the `script.user.scripts` table. If there is an existing anonymous script, it is replaced by the new one.

When the instrument is turned off, everything in the run-time environment is deleted, including the scripts and global variables.

See the figure below to see how the scripts, global variables, and `script.user.scripts` table interrelate.

Figure 112: Global variables and scripts in the runtime environment



Create a script using the script.new() command

Use the `script.new()` function to copy an existing script from the local node to a remote node. This enables parallel script execution.

You can create a script with the `script.new()` function using the command:

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

Where:

scriptVar	= Name of the variable created when the script is loaded into the run-time environment
code	= Content of the script
name	= Name that is added to the `script.user.scripts` table

For example, to set up a two-second beep, you can send the command:

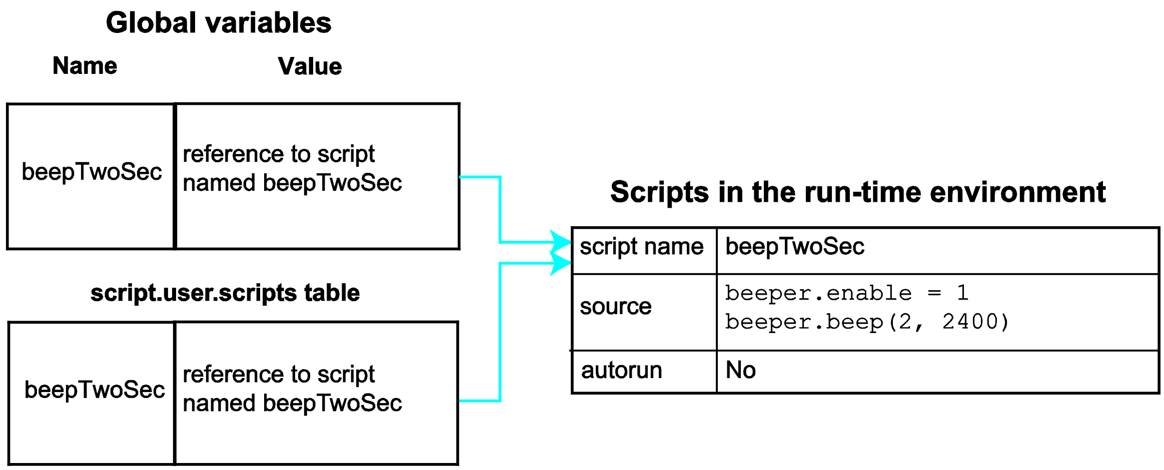
```
beepTwoSec = script.new("beeper.enable = 1 beeper.beep(2, 2400)", "beepTwoSec")
```

To run the new script, send the command:

```
beepTwoSec()
```

When you add `beepTwoSec`, the global variable and `script.user.scripts` table entries are made to the run-time environment as shown in the following figure.

Figure 113: Runtime environment after creating a script



Create an unnamed script using script.new()

NOTE

Unnamed scripts are not available from the front-panel display of the instrument. Only the anonymous script and named scripts are available from the front-panel display.

When you create a script using `script.new()`, if you do not include *name*, the script is added to the run-time environment as an unnamed script. The `script.new()` function returns the script. You can assign it to a global variable, a local variable, or ignore the return value. A global variable is not automatically created.

For example, send the following command:

```
hello = script.new('display.clear() display.settext("hello")')
```

A script is created in the run-time environment and a global variable is created that references the script.

To run the script, send the command:

```
hello()
```

Figure 114: Create an unnamed script

Global variables

Name	**Value**
hello	reference to unnamed script

Scripts in the runtime environment

script name	
source	`display.clear()` `display.settext("hello")`
Autorun	No

A script will become unnamed if you create a new script with the same name. In this circumstance, the name of the script in the `script.user.scripts` table is set to an empty string before it is replaced by the new script.

For example, if `beepTwoSec` already exists in the `script.user.scripts` table and you sent:

```
beepTwoSec1200 = script.new("beeper.enable = 1 beeper.beep(2, 1200)", "beepTwoSec")
```

The following actions occur:

- `beepTwoSec1200` is added as a global variable.
- The script that was in the run-time environment as `beepTwoSec` is changed to an unnamed script (the name attribute is set to an empty string).
- The global variable `beepTwoSec` remains in the run-time environment unchanged (it points to the now unnamed script).
- A new script named `beepTwoSec` is added to the run-time environment.

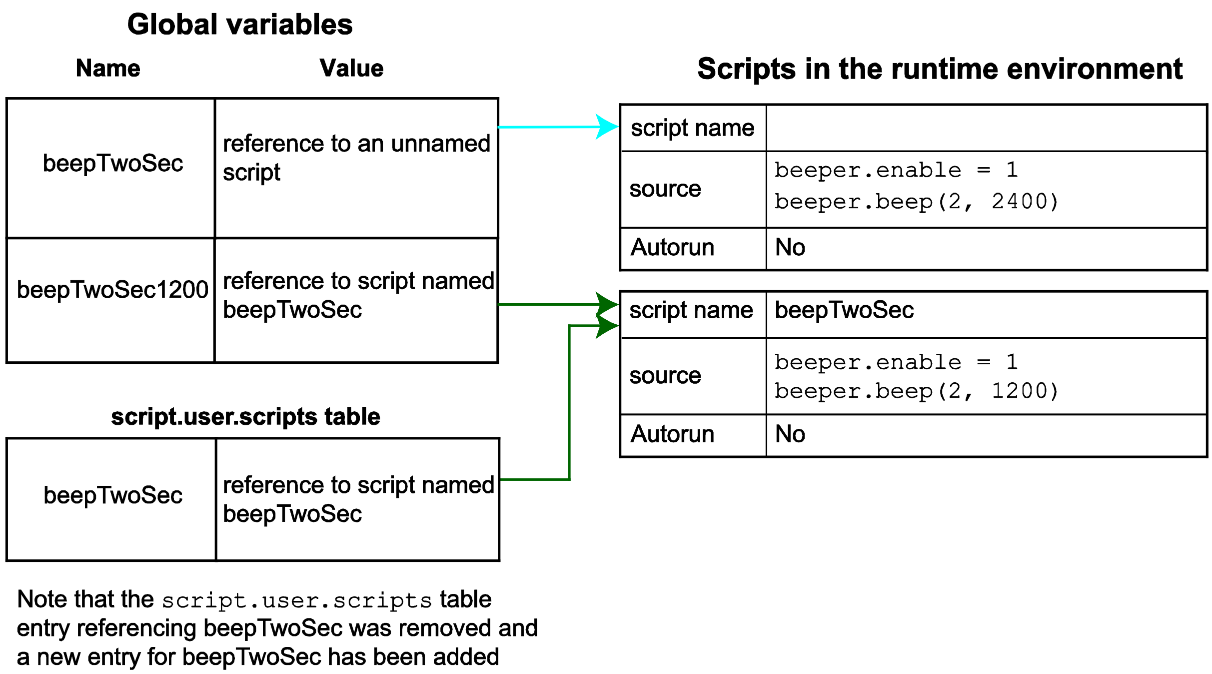
In this example, you can access the new script by sending either of the following commands:

```
beepTwoSec1200()
script.user.scripts.beepTwoSec()
```

To access the unnamed script, you can send the command:

```
beepTwoSec()
```

Figure 115: Create a new script with the name of an existing script



Restore a script to the run-time environment

You can retrieve a script that was removed from the run-time environment but is still saved in nonvolatile memory.

To restore a script from nonvolatile memory into the run-time environment, you can use `script.restore("`*`scriptName`*`")`, where *`scriptName`* is the user-defined name of the script to be restored.

For example, to restore a user script named `"test9"` from nonvolatile memory:

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

Rename a script

You can rename a script. You might want to rename a script if you need to name another script the same name as the existing script. You could also rename an existing script to be the autoexec script.

To change the name of a script, use the command:

```
scriptVar.name = "renamedScript"
```

Where:

`scriptVar`	=	The global variable name
`"`*`renamedScript`*`"`	=	The new name of the user script that was referenced by the *`scriptVar`* global variable

After changing the name, you need to save the original script to save the change to the name attribute.

For example:

```
beepTwoSec.name = "beep2sec"
beepTwoSec.save()
```

Run the `beep2sec` script using the following command:

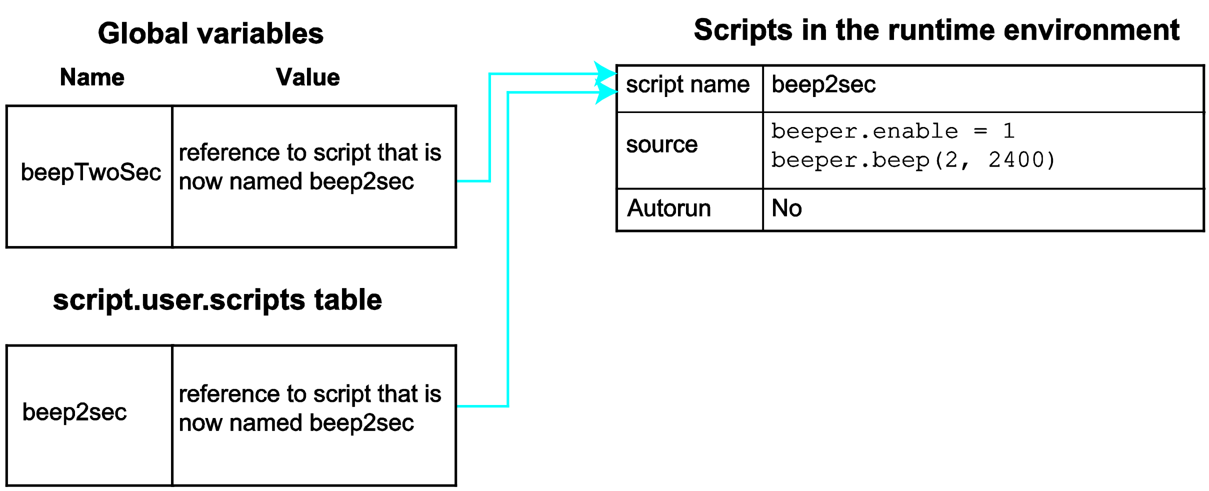
```
script.user.scripts.beep2sec()
```

NOTE

If the new name is the same as a name that is already used for a script, the name of the existing script is removed and that script becomes unnamed. This removes the existing script if there are no other variables that reference the previous script. If variables do reference the existing script, the references remain intact.

Changing the name of a script does not change the name of any variables that reference that script. After changing the name, the script is located in the `script.user.scripts` table under its new name.

Figure 116: Rename script



For example, to change the name of the script named `test2` to be `autoexec`:

```
test2.name = "autoexec"
test2.save()
```

The `autoexec` script runs automatically when the instrument is turned on. It runs after all the scripts have loaded and any scripts marked as autorun have run.

NOTE

You can also use the `script.new()` and the *`scriptVar`*`.source` attribute commands to create a script with a new name. For example, if you had an existing script named `test1`, you could create a new script named `test2` by sending the command:

```
test2 = script.new(test1.source, "test2")
```

See script.new() (on page 8-358).

Delete user scripts from the instrument

In most circumstances, you can delete a script using `script.delete()` (as described in Delete user scripts (on page 7-10)), and then turn the instrument off and back on again. However, if you cannot turn the instrument off, you can use the following steps to completely remove a script from the instrument.

When you completely remove a script, you delete all references to the script from the run-time environment, the `script.user.scripts` table, and nonvolatile memory.

To completely remove a script:

1. **Remove the script from the run-time environment.** Set any variables that refer to the script to `nil` or assign the variables a different value. For example, to remove the script `"beepTwoSec"` from the run-time environment, send the following code:
   ```
   beepTwoSec = nil
   ```
2. **Remove the script from the script.user.scripts table.** Set the name attribute to an empty string (`""`). This makes the script nameless, but does not make the script become the anonymous script. For example, to remove the script named `"beepTwoSec"`, send the following code:
   ```
   script.user.scripts.beepTwoSec.name = ""
   ```
3. **Remove the script from nonvolatile memory.** To delete the script from nonvolatile memory, send the command:
   ```
   script.delete("name")
   ```
   Where *name* is the name that the script was saved as. For example, to delete `"beepTwoSec"`, you would send:
   ```
   script.delete("beepTwoSec")
   ```

Memory considerations for the run-time environment

The run-time environment has a fixed amount of memory for storing user scripts channel patterns, DMM configurations, and other run-time information.

You can check the amount of memory in the instrument using the `memory.used()` and `memory.available()`functions. These functions return the percentage of memory that is used or available. When you send this command, memory used or available is returned as a comma-delimited string with percentages for used memory.

The format is `systemMemory, scriptMemory, patternMemory`, `configurationMemory`, where:

- `systemMemory`: The percentage of memory used or available in the instrument
- `scriptMemory`: The percentage of memory used or available in the instrument to store user scripts
- `patternMemory`: The percentage of memory used or available in the instrument to store channel patterns
- `configurationMemory`: The percentage of memory available to store DMM configurations.

For example, if you send the command:

```
MemUsed = memory.used()
print(MemUsed)
```

You will get back a value such as:

`69.14, 0.16, 12.74, 15.35`

Where:

- `69.14` is the percentage of memory used in the instrument
- `0.16` is the percentage used for script storage
- `12.74` is the percentage used for channel pattern storage
- `15.35` is the percentage used for DMM configuration storage

See memory.available() (on page 8-302) and memory.used() (on page 8-303) for more detail on using these functions.

Some suggestions for increasing the available memory:

- Turn the instrument off and on. This deletes scripts that have not been saved and reloads only scripts that have been stored in nonvolatile memory.
- Remove unneeded scripts from nonvolatile memory. Scripts are loaded from nonvolatile memory into the run-time environment when the instrument is turned on. See Delete user scripts from the instrument (on page 7-43).
- Reduce the number of TSP-Link® nodes.
- Delete unneeded channel patterns (this affects only pattern memory, not instrument memory). See Channel patterns (on page 2-96).
- Delete unneeded DMM configurations (this affects only configuration memory, not instrument memory. See Save DMM configurations (on page 4-7).
- Delete unneeded global variables from the run-time environment by setting them to `nil`.
- Set the source attribute of all scripts to `nil`.
- Adjust the `collectgarbage()` settings in Lua. See Lua memory management (on page 7-30) for information.
- Review scripts to optimize 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.

The example below shows an example that optimizes a channel pattern that consists of five channels.

Example

String concatenation lines	Optimized with the table.concat function
`ch1 = "" .. 5 * 1000 + 15 .. ","` `ch2 = "" .. 5 * 1000 + 25 .. ","` `ch3 = "" .. 5 * 1000 + 35 .. ","` `ch4 = "" .. 5 * 1000 + 915 .. ","` `ch5 = "" .. 5 * 1000 + 925` `testPattern = ch1 .. ch2 .. ch3 .. ch4 .. ch5` `print(testPattern)`	`testTable = { }` `testTable[1] = "5015,"` `testTable[2] = "5025,"` `testTable[3] = "5035,"` `testTable[4] = "5915,"` `testTable[5] = "5925"` `testPattern = table.concat(testTable)` `print(testPattern)`
The output is: `5015,5025,5035,5915,5925`	The output is: `5015,5025,5035,5915,5925`

CAUTION

If the instrument encounters memory allocation errors when memory used is above 95 percent, the state of the instrument cannot be guaranteed. After attempting to save any important data, it is recommended that you turn off power to the instrument and turn it back on to return the instrument to a known state. Cycling power resets the run-time environment. Unsaved scripts and channel patterns will be lost.

TSP-Link system expansion interface

Overview

The TSP-Link® expansion interface allows the Series 3700A instrument to communicate with other Test Script Processor (TSP®) enabled instruments. The test system can be expanded to include up to 32 TSP-Link enabled instruments.

CAUTION

Combining two Series 3700A instruments to achieve greater currents in both source voltage and source current applications requires specific precautions, including configuration settings. Make sure that you adequately understand the risks involved and the measures needed to accommodate the combination of two Series 3700A instruments. To prevent damage to the Series 3700A, connected instruments, and the device under test, make sure proper procedures are used. For further information, visit the Keithley Instruments website (*http://www.keithley.com*) for application notes on combining two Series 3700A channels.

Master and subordinates

In a TSP-Link® system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link® system can control the other nodes (subordinates) in the system.

When any node transitions from local operation to remote operation, it becomes the master of the system. All other nodes also transition to remote operation and become its subordinates. When any node transitions from remote operation to local, all other nodes also transition to local operation, and the master/subordinate relationship between nodes is dissolved.

In a TSP-Link® system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link® system can control the other nodes (subordinates) in the system.

When any node transitions from local operation to remote operation, it becomes the master of the system. All other nodes also transition to remote operation and become its subordinates. When any node transitions from remote operation to local, all other nodes also transition to local operation, and the master/subordinate relationship between nodes is dissolved.

The expanded system can be stand-alone or computer-based.

Stand-alone system: 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 (REM indicators turn on). The node running the script becomes the master and can control all of the other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM indicators turn off), and the master/subordinate relationship between nodes is dissolved.

Computer-based system: You can use a computer and a LAN, GPIB, or RS-232 interface to any 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 (REM indicators turn on). The node that receives the command becomes the master and can control all of the other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation.

TSP-Link system

You can use the TSP-Link® expansion interface to expand your test system to include up to 32 addressable TSP® enabled instruments that use the TSP-LINK®. The expanded system can be stand-alone or computer-based.

Stand-alone system: 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 (REM indicators turn on). The node running the script becomes the master and can control all of the other nodes, which become its subordinates. When the script is finished running, all the nodes in the system return to local operation (REM indicators turn off), and the master/subordinate relationship between nodes is dissolved.

Computer-based system: You can use a computer and a LAN, GPIB, or RS-232 interface to any 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 (REM indicators turn on). The node that receives the command becomes the master and can control all of the other nodes, which become its subordinates. In a computer-based system, the master/subordinate relationship between nodes can only be dissolved by performing an abort operation.

TSP-Link nodes

Each instrument (node) attached to the TSP-Link® network must be identified by assigning it a unique TSP-Link node number.

Commands for remote nodes are stored in the `node` table. An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node.

All TSP-accessible remote commands can be accessed as elements of the specific node. The following attributes are examples of items you can access:

- `node[N].model`: The product model number string of the node.
- `node[N].revision`: The product revision string of the node.
- `node[N].serialno`: The product serial number string of the node.

You do not need to know the node number of the node that is running a script. The variable `localnode` is an alias for the node entry of the node where the script is running. For example, if a script is running on node 5, you can use the global variable `localnode` as an alias for `node[5]`. With this in mind, to access the product model number for this example, use `localnode.model`.

Connections

Connections for an expanded system are shown in the following figure. As shown, one instrument is optionally connected to the computer using the GPIB, LAN, or RS-232 interface. Details about these computer communication connections are described in Remote communication interfaces (on page 2-53).

All the instruments in the system are connected in a sequence (daisy-chained) using LAN crossover cables.

Initialization

Before a TSP-Link® system can be used, it must be initialized. For initialization to succeed, each instrument in a TSP-Link system must be assigned a different node number.

Assigning node numbers

At the factory, each Series 3700A instrument is assigned as node 1. 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 a Series 3700A using the front panel or by using a remote command. Note that there can only be 32 physical nodes, but you can assign node numbers from 1 to 64.

To assign a node number from the front panel of the instrument:

1. Press the **MENU** key, then select **TSPLINK > NODE**.
2. Press the navigation wheel ◷ and select the desired number.
3. Press the **ENTER** key to save the node number.

To assign a node number using a remote command:

Set the `tsplink.node` attribute of the instrument:

```
tsplink.node = N
```

Where: *N* = 1 to 64

To determine the node number of an instrument, you can read the `tsplink.node` attribute by sending the following command:

```
print(tsplink.node)
```

The above `print` command outputs the node number. For example, if the node number is 1, a `1` is displayed.

Resetting the TSP-Link network

After all the node numbers are set, you must initialize the system by performing a TSP-Link® network reset.

NOTE

If you change the system configuration after initialization, you must reinitialize the system by performing a TSP-Link network reset. Changes that require that you reinitialize the TSP-Link network include turning off power or rebooting any instrument in the system, or rearranging or disconnecting the TSP-Link cable connections between instruments.

Front panel operation

To reset the TSP-Link® network from the front panel:

1. Power on all instruments connected to the TSP-Link network.
2. Press the **MENU** key, select **TSPLINK**, and then press the **ENTER** key.
3. Turn the navigation wheel ◷ to select **RESET**, and then press the **ENTER** key.

Remote programming

The commands associated with the TSP-Link® system reset are listed in the following table.

TSP-Link reset commands

Command	Description
`tsplink.reset()`	Initializes the TSP-Link network
`tsplink.state`	Reads the state of the TSP-Link network: • "`online`" if the most recent TSP-Link reset was successful • "`offline`" if the reset operation failed

An attempted TSP-Link reset operation will fail if any of the following conditions are true:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument performing the reset (only if the expected number of nodes was not provided in the reset call)
- One or more of the instruments in the system is turned off
- If the actual number of nodes is less than the expected number

The programming example below illustrates a TSP-Link reset operation and displays its state:

```
tsplink.reset()
print(tsplink.state)
```

If the reset operation is successful, `online` is output to indicate that communications with all nodes have been established.

Using the expanded system

Accessing nodes

You can access all the remote commands for a specific node by adding `node[N].` to the beginning of the remote command, where *N* is the node number.

The variable `localnode` is an alias for `node[N]`, where *N* is the node number of the node on which the code is running. For example, if node 1 is running the code, `localnode` can be used instead of `node[1]`.

The following programming examples illustrate how to access instruments in the TSP-Link system (shown in TSP-Link connections):

- You can use any one of the following three commands to reset all channels of node 1 (which, in this example, is the master). The other nodes in the system are not affected.

```
channel.reset("allslots")
localnode.channel.reset("allslots")
node[1].channel.reset("allslots")
```

- The following command will reset all channels of node 4, which is a subordinate. The other nodes are not affected.

```
node[4].channel.reset("allslots")
```

Using the reset() command

Most TSP-Link® system operations target a single node in the system, but the `reset()` command affects the system as a whole by resetting all nodes to their default settings:

```
-- Reset all nodes in a TSP-Link system to their default state.
reset()
```

NOTE

Using the `reset()` command in a TSP-Link network differs from using the `tsplink.reset()` command. The `tsplink.reset()` command reinitializes the TSP-Link network, but does not change the state of the 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 node 1 only.
localnode.reset()
-- Reset node 4 only.
node[4].reset()
```

A TSP-Link® reset command populates the node table. Each instrument in the system corresponds to an entry in this table. Each entry is indexed by the node number of the instrument. The variable `node[N]` (where *N* is the node number) is used to access any node in the system. For example, node 1 is represented as entry `node[1]` in the node table.

Each of these entries is a table holding all the remote commands shared by the corresponding instrument. Source-measure unit (SMU) A on node 1, therefore, could be accessed as `node[1].smua`.

Using the abort command

An `abort` command terminates an executing script and returns all nodes to local operation (REM indicators turn off). This dissolves the master/subordinate relationships between nodes. To invoke an abort operation, either send an `abort` command to a specific node or press the EXIT (LOCAL) key on any node in the system.

You can also perform an abort operation by pressing the OUTPUT ON/OFF control on any node. The results are the same as above, with the addition that all outputs in the system are turned off.

Triggering with TSP-Link

The TSP-Link® expansion interface has three synchronization lines that function similarly to the digital I/O synchronization lines. See Digital I/O (on page 3-43) and Triggering for more information.

TSP advanced features

Use the Test Script Processor (TSP®) scripting engine's advanced features to:

- Run test scripts simultaneously
- Manage resources allocated to test scripts that are running simultaneously
- Use the data queue to facilitate real-time communication between nodes on the TSP-Link® network

When test scripts are run simultaneously, it improves functional testing, provides higher throughput, and expands system flexibility.

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

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

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

Another method you can use to run simultaneous test scripts is to use groups with a single TSP-Link network. Each group on the TSP-Link network can run a test while other groups are running different tests.

A group consists of one or more nodes with the same group number. The following figure displays a single TSP-Link network with groups. This method requires one TSP-Link network and a single GPIB connection to the computer.

The following table shows an example of the functions of 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.

Using groups to manage nodes on TSP-Link network

The primary purpose of groups is to allow each group to run a different test script simultaneously.

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).

Master node overview

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to group 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

The master node is always the node that coordinates activity on the TSP-Link network.

The master node:

- Is the only node that can use the `execute()` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation (a command that continues to operate after the command that initiated it has finished running)
- Can execute the `waitcomplete()` command to wait for the group to which the master node belongs; to wait for another group; or to wait for all nodes on the TSP-Link network to complete overlapped operations (overlapped commands allow the execution of subsequent commands while device operations of the overlapped command are still in progress)

Group leader overview

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

The group leader:

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

Assigning groups

Group numbers can range from zero (0) to 64. The default group number is 0. You can change the group number at any time. You can also add or remove a node to or from a group at any time.

Each time the node's power is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
-- Assign node 3 to group 1.
node[3].tsplink.group = 1
```

Running simultaneous test scripts

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

Use the following code to send the `execute()` command for a remote node. The *`N`* parameter represents the node number that runs the test script (replace *`N`* with the node number).

To set the global variable "setpoint" on node N to 2.5:

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

The following code demonstrates how to run a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run scriptVar on node N:

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

The programming example below demonstrates how to run a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

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

It is recommended that you copy large scripts to a remote node to improve system performance. See Copying test scripts across the TSP-Link network (on page 7-54) for more information.

Coordinating overlapped operations in remote groups

All overlapped operations on all nodes in a group must have completed before the master node can send a command to the group. If you send a command to a node in a remote group when an overlapped operation is running on any node in that group, errors will occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, see waitcomplete() (on page 8-466).

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()
```

NOTE

Presently, the Series 3700A has no overlapped commands implemented. However, other TSP-enabled products, such as the Series 2600A System SourceMeter® Instruments, have overlapped commands. Therefore, when the Series 3700A is a TSP master to a subordinate device with overlapped commands, use this function to wait until all overlapped operations are completed.

Using the data queue for real-time communication

Nodes that are running test scripts at the same time can store data in the data queue for real-time communication. Each instrument has an internal data queue that uses the first-in, first-out (FIFO) structure to store data. You can use the data queue to post numeric values, strings, and tables.

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link® network at any time

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

Tables in the data queue consume one entry. When a node stores a table in the data queue, a copy of the data in the table is made. When the data is retrieved from the data queue, a new table is created on the node that is retrieving the data. The new table contains a completely separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` commands in the TSP command reference (on page 8-1) for more information.

Copying test scripts across the TSP-Link network

To run a large script on a remote node, copy the test script to the remote node to increase the speed of test script initiation.

The code in the example below copies a test script across the TSP-Link® network, creating a copy of the script on the remote node with the same name.

```
-- Add the source code from the script
-- testScript to the data queue.
node[2].dataqueue.add(testScript.source)
-- Create a new script on the remote node
-- using the source code from testScript.
node[2].execute(testScript.name ..
   "= script.new(dataqueue.next(), [[" .. testScript.name .. "]])")
```

Removing stale values from the reading buffer cache

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

When you run Lua code remotely, it can cause reading buffer data that is held in the cache to become stale. If the values in the reading buffer change while the Lua code runs remotely, another node can hold stale values. Use the `clearcache()` command to clear the cache. For additional detail on the reading buffer cache commands, see bufferVar.cachemode (on page 8-20) and bufferVar.clearcache() (on page 8-23).

The following example code demonstrates how stale values occur and how to use the `clearcache()` command to clear the cache on node 2, which is part of group 7.

```
-- Create a reading buffer on a node in a remote group.
node[2].tsplink.group = 7
node[2].execute("rbremote = dmm.makebuffer(20) " ..
                "dmm.measure.count = 20 " ..
                "dmm.measure(rbremote)")
-- Create a variable on the local node to
-- access the reading buffer.
rblocal = node[2].getglobal("rbremote")
-- Access data from the reading buffer.
print(rblocal[1])
-- Run code on the remote node that updates the reading buffer.
node[2].execute("dmm.measure(rbremote)")
-- Use the clearcache command if the reading buffer contains cached data.
rblocal.clearcache()
-- If you do not use the clearcache command, the data buffer
-- values will never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values will print.
print(rblocal[1])
```

TSP-Net

Overview

The TSP-Net® library allows the Series 3700A to control LAN-enabled devices directly through its LAN port. This enables the Series 3700A to communicate directly with a device that is not TSP® enabled without the use of a controlling computer.

TSP-Net capabilities

The TSP-Net library permits the Series 3700A to control a remote instrument through the LAN port for both Test Script Processor (TSP®) and non-TSP instruments. 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.

You can use TSP-Net commands to communicate with any ethernet-enabled instrument. However, 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 Series 3700A can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The Series 3700A 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 simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the Series 3700A.

Using TSP-Net with any Ethernet-enabled device

NOTE

Refer to TSP commands (on page 8-10) for more details about the commands presented in this section.

To communicate to a remote ethernet-enabled device from the Series 3700A, perform the following steps:

1. Connect to the remote device through the LAN port. If you are connecting:
   - Directly from the Series 3700A to an ethernet-enabled device: Use an ethernet crossover cable.
   - The Series 3700A to any other device on the LAN: Use a straight-through ethernet cable and a hub.
1. Establish a new connection to a remote device at a specific IP address using `tspnet.connect()`.
2. If the device is not TSP-enabled, you must also provide the port number. If not, the Series 3700A assumes the remote device is TSP-enabled and enables TSP prompts and error handling.

   If the Series 3700A is not able to make a connection to the remote device, it generates a timeout error. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 seconds.

NOTE

Set `tspnet.tsp.abortonconnect` to 1 to abort any script currently running on a remote TSP device.

3. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. `tspnet.write()` sends strings to the device exactly as indicated, and you must supply any needed termination characters or other lines. Use `tspnet.termination()` to specify the termination character. If you use `tspnet.execute()` (on page 8-453) instead, the Series 3700A appends termination characters to all strings sent to the command.
4. Retrieve responses from the remote device using `tspnet.read()`. The Series 3700A suspends operation until data is available or a timeout error is generated. You can check if data is available from the remote device using `tspnet.readavailable()`.

Disconnect from the remote device using `tspnet.disconnect()`. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not Test Script Processor (TSP®) enabled, and send and receive data from this device:

```
-- Disconnect all existing TSP-Net connections.
tspnet.reset()
-- Set tspnet timeout to 5 seconds.
tspnet.timeout = 5
-- Establish connection to another device with
-- IP address 192.168.1.51 at port 1394.
id_instr = tspnet.connect("192.168.1.51",1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set termination character to CRLF. You must do this
-- on a per connection basis after connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device
tspnet.write(id_instr, "*idn?" .. "\r\n")
-- Read the data available, then prints it.
print("instrument write/read returns:: , tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

Using TSP-Net with any ethernet-enabled instrument

NOTE

Refer to TSP command reference (on page 8-1) for details about the commands presented in this section.

The Series 3700A LAN connection is auto-sensing (Auto-MDIX), so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the Series 3700A to an ethernet device or to a hub.

To set up communication to a remote ethernet-enabled instrument that is TSP® enabled:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

```
tspnet.tsp.abortonconnect = 1
```

   If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

```
connectionID = tspnet.connect(ipAddress)
```

   Where:

   - `connectionID` is the connection ID that will be used as a handle in all other TSP-Net function calls.
   - `ipAddress` is the IP address of the remote instrument.

See tspnet.connect() (on page 8-451) for additional detail.

To set up communication to a remote ethernet-enabled device that is not TSP enabled:

1. Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

   Where:

   - `connectionID` is the connection ID that will be used as a handle in all other `tspnet` function calls.
   - `ipAddress` is the IP address of the remote device.
   - `portNumber` is the port number of the remote device.
   - `initString` is the initialization string that is to be sent to `ipAddress`.

See tspnet.connect() (on page 8-451) for additional detail.

To communicate to a remote ethernet device from the Series 3700A:

1. Connect to the remote device using one of the above procedures. If the Series 3700A cannot make a connection to the remote device, it generates a timeout error. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 seconds.
2. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. If you use:
   - `tspnet.write()`: Strings are sent to the device exactly as indicated, and you must supply any needed termination characters.
   - `tspnet.execute()`: The Series 3700A appends termination characters to all strings that are sent. Use `tspnet.termination()` to specify the termination character.

1. To retrieve responses from the remote instrument, use `tspnet.read()`. The Series 3700A suspends operation until the remote device responds or a timeout error is generated. To check if data is available from the remote instrument, use `tspnet.readavailable()`.
2. Disconnect from the remote device using the `tspnet.disconnect()` function. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not TSP® enabled, and send and receive data from this device:

```
-- Disconnect all existing TSP-Net connections.
tspnet.reset()
-- Set tspnet timeout to 5 seconds.
tspnet.timeout = 5
--  Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set the termination character to CRLF. You must do this
-- for each connection after the connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "*idn?" .. "\r\n")
-- Read the data available, then print it.
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

TSP-Net compared to TSP-Link to communicate with TSP-enabled devices

The TSP-Link® network interface is the preferred communication method for most applications where communication occurs between the Series 3700A and another TSP-enabled instrument.

One of the advantages of using the TSP-Link network interface is that TSP-Link connections have three synchronization lines that are available to each device on the TSP-Link network. You can use any one of the synchronization lines to perform hardware triggering between devices on the TSP-Link network. Refer to Hardware trigger modes (on page 3-10) for details.

However, if the distance between the Series 3700A and the TSP-enabled device is longer than 15 feet, use TSP-Net commands.

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

tspnet.clear() (on page 8-451)
tspnet.connect() (on page 8-451)
tspnet.disconnect() (on page 8-452)
tspnet.execute() (on page 8-453)
tspnet.idn() (on page 8-454)
tspnet.read() (on page 8-455)
tspnet.readavailable() (on page 8-456)
tspnet.reset() (on page 8-457)
tspnet.termination() (on page 8-457)
tspnet.timeout (on page 8-458)
tspnet.write() (on page 8-461)

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

tspnet.tsp.abort() (on page 8-458)
tspnet.tsp.abortonconnect (on page 8-459)
tspnet.tsp.rbtablecopy() (on page 8-460)
tspnet.tsp.runscript() (on page 8-461)

Example: Using tspnet commands

```
function telnetConnect(ipAddress, userName, password)
   -- Connect through Telnet to a computer.
   id = tspnet.connect(ipAddress, 23, "")
   -- Read the title and login prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the login name.
   tspnet.write(id, userName .. "\r\n")
   -- Read the login echo and password prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the password information.
   tspnet.write(id, password .. "\r\n")
   -- Read the telnet banner from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
end

function test_tspnet ()
   tspnet.reset()
   -- Connect to a computer using Telnet.
   telnetConnect("192.0.2.1", "my_username", "my_password")
   -- Read the prompt back from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   -- Change directory and read the prompt back from the computer.
   tspnet.write(id, "cd c:\\\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Make a directory and read the prompt back from the computer.
   tspnet.write(id, "mkdir TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Change to the newly created directory.
   tspnet.write(id, "cd c:\\TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- if you have data print it to the file.
   -- 11.2 is an example of data collected.
   cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
   tspnet.write(id, cmd)
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   tspnet.disconnect(id)
end
test_tspnet()
```

Section 8

TSP command reference

In this section:

Command programming notes

Placeholder text

This manual uses italicized text to represent the parts of remote commands that must be replaced by user specified values. The following examples show typical uses of italicized text:

Example 1:

`gpib.address = `*`address`*

Where:

`address` is an integer (0 to 30) that you specify. For example, to set this attribute to 15 you would send:

```
gpib.address = 15
```

Example 2:

`digio.trigger[`*`N`*`].assert()`

Where:

`N` is an integer (1 to 14) that you specify. For example, to assert trigger line 7 you would send:

```
digio.trigger[7].assert()
```

To assert a trigger line with a variable as the integer, you would send:

```
triggerline = 7
digio.trigger[triggerline].assert()
```

Syntax rules

The following table lists syntax requirements to build well-formed instrument control commands.

Syntax rules for instrument commands

Syntax rule	Details	Examples
Case sensitivity: Instrument commands are case sensitive. Match the case shown in the command reference descriptions.	Function and attribute names should be in lowercase characters.	An example of the `scriptVar.save()` function (where `test8` is the name of the script): `test8.save()`
	Parameters can use a combination of lowercase and uppercase characters. Attribute constants use uppercase characters	In the command below, which sets the format of data transmitted from the instrument to double-precision floating point, `format.REAL64` is the attribute constant and `format.data` is the attribute command: `format.data = format.REAL64`
White space: Not required in a function.	Functions can be sent with or without white spaces.	The following functions, which set digital I/O line 3 low, are equivalent: `digio.writebit(3,0)` `digio.writebit (3, 0)`
Function parameters: All functions are required to have a set of parentheses () immediately following the function.	You can specify the function parameters by placing them between the parentheses. Note that the parentheses are required even when there are no parameters specified.	The following function specifies all overlapped commands in the nodes in group G that must complete before commands from other groups can execute: `waitcomplete(G)` The command below reads the value of the local time zone (no parameters are needed): `timezone = gettimezone()`
Multiple parameters: Must be separated by commas (,).	Some commands require multiple parameters, which must be separated by commas (`,`).	This command sets the beeper to emit a double-beep at 2400 Hz, with a beep sequence of 0.5 seconds on, 0.25 seconds off, and then 0.5 seconds on: `beeper.beep(0.5, 2400)` `delay(0.250)` `beeper.beep(0.5, 2400)`
Parameter range: Range values must be separated with a colon (:).	Place a colon (:) between two values to specify a range in a parameter.	The command below replaces the active scan list with an empty scan list, and then adds channels 1 through 10 on slot 1: `scan.create("1001:1010")`

Logical instruments

You would normally refer to all instrumentation in one enclosure or node as a single instrument. In the context of Test Script Processor (TSP®) scripting engine and instrument commands, it is useful to think of each individual subdivision in an enclosure, such as a card slot or the channels, as a stand-alone instrument. To avoid confusion, all subdivisions of the instrumentation in an enclosure are referred to as "logical instruments."

Each logical instrument is given a unique identifier in a system. These identifiers are used as part of all commands that control a given logical instrument.

The logical instruments are:

- beeper
- bit
- channel
- dataqueue
- digio
- display
- dmm
- errorqueue
- eventlog
- format
- fs
- gpib
- io
- lan
- memory
- ptp
- scan
- schedule
- setup
- slot
- status
- timer
- trigger
- tsplink
- tspnet
- upgrade
- userstring

NOTE

Do not create variable names that are the same as names of logical instruments. Doing so will result in the loss of use of the logical instrument and its associated commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you turn off the power to the instrument, and then turn it on again.

Using channel.*() commands

Unless otherwise noted, `channel.*()` remote commands use the channel list syntax described below.

- The channel list is specified according to the syntax presented in the channel list legend. Not all remote commands support the fully described syntax. Any exclusions are noted in the documentation for a specific command.
- There are five different types of channels available on the supported Series 3700A cards. These include switch (or relay), backplane, totalizer, DAC, and digital I/O. Even though the channels are specified in an identical manner, not all remote commands act on all channel types. The description of each remote command provides more information.
- When acting on a range of channels is necessary or more convenient, use the ":" notation. For example, to specify channels 1 through 20 on slot 4, use `4001:4020`.

```
print(channel.getlabel("4001:4020"))
```

- When acting on an entire slot is necessary or more convenient, use the `slot`*X* notation. For example, to specify all channels on slot 4, use `slot4`.

```
print(channel.getlabel("slot4"))
```

- When acting on an entire instrument is necessary or more convenient, use the `allslots` notation. For example, to specify channels on all slots (1 through 6), use `allslots.`

```
print(channel.getlabel("allslots"))
```

- When a range (including `slot`*X* and `allslots` notation) includes mixed channel types, the invalid channel types are ignored. If an invalid channel type is individually specified, then an error is generated.

The following errors can occur because of invalid channel list syntax or specification.

Error Message	Description
`invalid specified channel`	The channel is specified with the correct syntax, but does not exist on the card.
`invalid character in channel list`	The channel list contains an invalid character or syntax sequence.
`invalid slot in channel list`	The slot specified in the channel list is empty.
`invalid channel type in channel list`	The channel is specified with the correct syntax, but the channel type is not supported by the specified remote command.
`no valid channels in channel list`	After processing, no valid channels remain in the command to act upon.
`invalid label or pattern name`	A string was found in the channel list that does not specify any known label or pattern name.
`no patterns accepted`	A pattern was specified for a remote command that does not support patterns as input.
`no multiple channels accepted`	Multiple channels were specified for a remote command that acts only on a single channel.
`no range specifier accepted`	A range was specified for a remote command that does not support a range as input.
`no slot specifier accepted`	An entire slot was specified using `slotX` (for example, `slot1`) for a remote command that does not support `slotX` as input.
`no all slots specifier accepted`	All slots were specified using `allslots` for a remote command that does not support `allslots` as input.
`no labels accepted`	A label was specified for a remote command that does not support labels as input.
`no paired channels accepted`	A channel was specified for a remote command that does not act on paired channels.
`no single channels accepted`	A single channel was specified for a remote command that only supports acting on groups of channels.
`no multiple specifiers accepted`	Multiple descriptions were specified for a remote command that does not support multiple descriptions in a list.
`channels all must be of same type`	The provided channel list contains channels of various channel types, but the remote command supports only channel lists that contain a single, consistent channel type.
`forbidden channel`	The channel specified is forbidden to be closed.

Time and date values

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

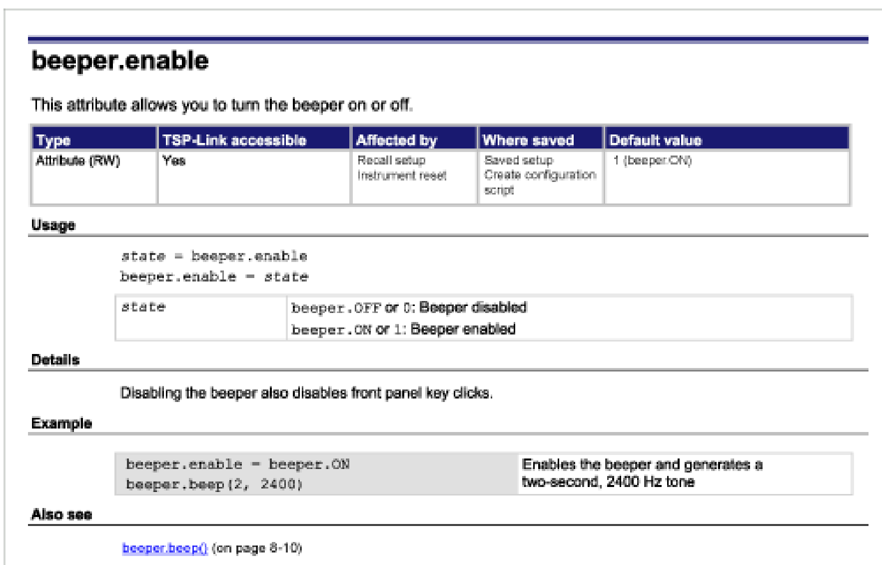
Time and date values are represented as the number of seconds since some base. Representing time as a number of seconds is referred to as "standard time format." The three time bases used for the Series 3700A are:

- **UTC 12:00 am Jan 1, 1970.** Some examples of UTC time are reading buffer seconds, adjustment dates, and the value returned by `os.time()`.
- **Instrument on.** References time to when the instrument was turned on. The value returned by `os.clock()` is referenced to the turn-on time.
- **Event.** Time referenced to an event, such as the first reading stored in a reading buffer.

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 117: Example instrument command description



beeper.enable

This attribute allows you to turn the beeper on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Recall setup Instrument reset	Saved setup Create configuration script	1 (beeper.ON)

Usage

```
state = beeper.enable
beeper.enable = state
```

state	`beeper.OFF` or `0`: Beeper disabled `beeper.ON` or `1`: Beeper enabled

Details

Disabling the beeper also disables front panel key clicks.

Example

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

Also see

beeper.beep() (on page 8-10)

The subsections contain information about the command. The subsections are:

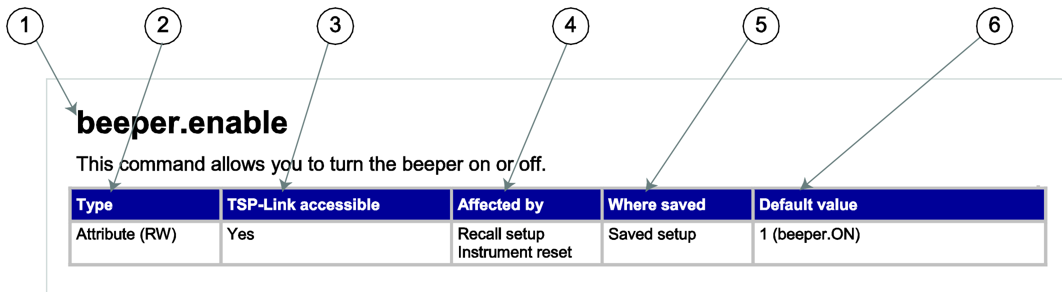
- Command name and summary table
- Usage
- Details
- Example
- Also see

The content of each of these subsections is described in the following topics.

Command name and standard parameters summary

Each instrument command description starts with the command name, followed by a table with relevant information for each command. Definitions for the numbered items in the figure below are listed following the figure.

Figure 118: Command name and summary table



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:
   - **Function**. Function-based commands control actions or activities, but are not always directly related to instrument operation. Function names are always followed by a set of parentheses, for example, `digio.writeport(15)`. If the function does not need a parameter, the parentheses set remains empty, for example, `exit()`.
   - **Attribute (R), (RW),** or **(W)**. Attribute-based commands set or read the characteristics of an instrument feature or operation by defining a value. For example, a characteristic of a TSP-enabled instrument is the model number (`localnode.model`); another characteristic is the number of errors in the error queue (`errorqueue.count`). For many attributes, the defined value is a number or predefined constant. Attributes can be read-only (R), read-write (RW), or write-only (W), and can be used as a parameter of a function or assigned to another variable.
   - **Constant.** A constant command represents a fixed value when used in a script.

1. **TSP-Link accessible**. **Yes** or **No**; indicates whether or not the command can accessed through a TSP-Link network.
2. **Affected by**. Commands or actions that have a direct effect on the instrument command.
   - **DMM configuration recall**
   - **DMM function change**
   - **DMM reset**
   - **LAN reset**
   - **Recall setup**
   - **Reset:** Reset commands affect commands in different ways, depending on the type of reset. Types of reset include:

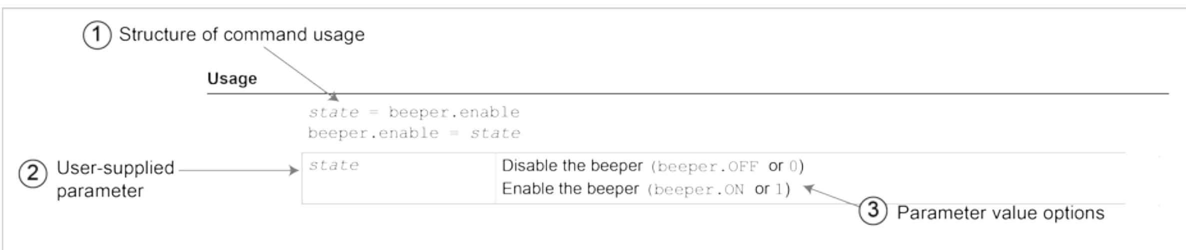
     - Channel reset
     - Digital I/O trigger reset
     - Instrument reset
     - Local node reset
     - Scan reset
     - Status reset
     - Trigger blender reset
     - Trigger timer reset
     - TSP-Link trigger reset

1. **Where saved**. Indicates where the command settings reside once they are used on an instrument. Options include:
   - **Create configuration script:** This command is saved as part of the configuration script if you save the current configuration into a script with the `createconfigscript()` command or the MENU > SCRIPT > CREATE-CONFIG option from the front panel.
   - **Not saved:** Command is not saved anywhere and must be typed each time you use it.
   - **Nonvolatile memory:** Storage area in the instrument where information is saved when the instrument is turned off.
   - **Saved setup**
   - **Setup:** Instrument settings are captured in an internal or external setup file to be recalled later.
1. **Default value:** Lists the default value or constant for the command. The parameter values are defined in the Usage or Details sections of the command description.

Command usage

The Usage section of the remote command listing shows how to properly structure the command. Each line in the Usage section is a separate variation of the command usage; all possible command usage options are shown here.

Figure 119: Command usage section



1 **Structure of command usage:** Shows how the parts of the command should be organized. If a parameter is shown to the left of the command, it is the return when you print the command. Information to the right are the parameters or other items you need to enter.

2. **User-supplied parameters:** Indicated by italics. For example, for the function `beeper.beep(`*`duration, frequency`*`)`, replace *`duration`* with the number of seconds and *`frequency`* with the frequency of the tone. `beeper.beep(2, 2400)` generates a two-second, 2400 Hz tone.

   Some commands have optional parameters. If there are optional parameters, they must be entered in the order presented in the Usage section. You cannot leave out any parameters that precede the optional parameter. Optional parameters are shown as separate lines in usage, presented in the required order with each valid permutation of the optional parameters.
   For example:
   `printbuffer(`*`startIndex`*, *`endIndex`*, *`buffer1`*`)`
   `printbuffer(`*`startIndex`*, *`endIndex`*, *`buffer1`*, *`buffer2`*`)`

3. **Parameter value options:** Descriptions of the options that are available for the user-defined parameter.

Command details

This section lists additional information you need to know to successfully use the remote commands.

Figure 120: Details section of command listing

Details

This attribute enables or disables the beeper.
Disabling the beeper also disables front-panel key clicks.

Example section

The Example section of the remote command description shows some simple examples of how the command can be used.

Figure 121: Code examples in command listings

1 Actual example code that you can copy from this table and paste into your own programming application.

2 Description of the code and what it does. This may also contain the output of the code.

Related commands and information

The Also See section of the remote command description lists additional commands that are related to the command being described.

Figure 122: Links to related commands and information

Also see

beeper.beep() (on page 7-8)

TSP commands

beeper.beep()

generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

duration	The amount of time to play the tone in seconds (0.1 to 100)
frequency	The frequency of the tone in Hertz (Hz)

Details

You can use the beeper of the Model 3706A to provide an audible signal at a specified frequency and time duration. For example, you can use the beeper to signal the end of a lengthy sweep.

The beeper will not sound if it is disabled. It can be disabled or enabled with the beeper enable command, or through the front panel.

Example

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

Also see

beeper.enable (on page 8-10)

beeper.enable

This command allows you to turn the beeper on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Recall setup Instrument reset	Saved setup Create configuration script	1 (beeper.ON)

Usage

```
state = beeper.enable
beeper.enable = state
```

state	Disable the beeper: `beeper.OFF` or `0` Enable the beeper: `beeper.ON` or `1`

Details

This command enables or disables the beeper. When enabled, a beep signals that a front-panel key has been pressed. Disabling the beeper also disables front-panel key clicks.

Example

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

Also see

beeper.beep() (on page 8-10)

bit.bitand()

This function performs a bitwise logical AND operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitand(value1, value2)
```

result	Result of the logical AND operation
value1	Operand for the logical AND operation
value2	Operand for the logical AND operation

Details

Any fractional parts of *value1* and *value2* are truncated to form integers. The returned *result* is also an integer.

Example

`testResult = bit.bitand(10, 9)` `print(testResult)`	Performs a logical AND operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 8 (binary 1000). Output: `8.00000e+00`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.bitor() (on page 8-11)
bit.bitxor() (on page 8-12)

bit.bitor()

This function performs a bitwise logical OR operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitor(value1, value2)
```

result	Result of the logical OR operation
value1	Operand for the logical OR operation
value2	Operand for the logical OR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

`testResult = bit.bitor(10, 9)` `print(testResult)`	Performs a bitwise logical OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 11 (binary 1011). Output: `1.10000e+01`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.bitand() (on page 8-11)
bit.bitxor() (on page 8-12)

bit.bitxor()

This function performs a bitwise logical XOR (exclusive OR) operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

result = bit.bitxor(*value1*, *value2*)

result	Result of the logical XOR operation
value1	Operand for the logical XOR operation
value2	Operand for the logical XOR operation

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

Example

`testResult = bit.bitxor(10, 9)` `print(testResult)`	Performs a logical XOR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 3 (binary 0011). Output: `3.00000e+00`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.bitand() (on page 8-11)
bit.bitor() (on page 8-11)

bit.clear()

This function clears a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`result = bit.clear(value, index)`

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to clear (1 to 32)

Details

Any fractional part of *value* is truncated to make it an integer. The returned *result* is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

`testResult = bit.clear(15, 2)` `print(testResult)`	The binary equivalent of decimal 15 is 1111. If you clear the bit at *index* position 2, the returned decimal value is 13 (binary 1101). Output: `1.30000e+01`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.get() (on page 8-13)
bit.set() (on page 8-15)
bit.test() (on page 8-16)
bit.toggle() (on page 8-17)

bit.get()

This function retrieves the weighted value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`result = bit.get(value, index)`

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get (1 to 32)

Details

This function returns the value of the bit in *value* at *index*. This is the same as returning *value* with all other bits set to zero (0).

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for the number is set to zero (0), the result will be zero (0).

Example

`testResult = bit.get(10, 4)` `print(testResult)`	The binary equivalent of decimal 10 is 1010. If you get the bit at index position 4, the returned decimal value is 8. Output: `8.00000e+00`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.clear() (on page 8-13)
bit.set() (on page 8-15)
bit.test() (on page 8-16)
bit.toggle() (on page 8-17)

bit.getfield()

This function returns a field of bits from the value starting at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.getfield(value, index, width)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get (1 to 32)
width	The number of bits to include in the field (1 to 32)

Details

A field of bits is a contiguous group of bits. This function retrieves a field of bits from *value* starting at *index*.

The *index* position is the least significant bit of the retrieved field. The number of bits to return is specified by *width*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Example

`myResult = bit.getfield(13, 2, 3)` `print(myResult)`	The binary equivalent of decimal 13 is 1101. The field at *index* position 2 and *width* 3 consists of the binary bits 110. The returned value is decimal 6 (binary 110). Output: `6.00000e+00`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.get() (on page 8-13)
bit.set() (on page 8-15)
bit.setfield() (on page 8-15)

bit.set()

This function sets a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.set(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to set (1 to 32)

Details

This function returns *result*, which is *value* with the indexed bit set. The *index* must be between 1 and 32.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Any fractional part of *value* is truncated to make it an integer.

Example

`testResult = bit.set(8, 3)` `print(testResult)`	The binary equivalent of decimal 8 is 1000. If the bit at *index* position 3 is set to 1, the returned value is decimal 12 (binary 1100). Output: `1.20000e+01`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.clear() (on page 8-13)
bit.get() (on page 8-13)
bit.getfield() (on page 8-14)
bit.setfield() (on page 8-15)
bit.test() (on page 8-16)
bit.toggle() (on page 8-17)

bit.setfield()

This function overwrites a bit field at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`result = bit.setfield(value, index, width, fieldValue)`

result	Result of the bit manipulation
value	Specified number
index	One-based bit position in *value* to set (1 to 32)
width	The number of bits to include in the field (1 to 32)
fieldValue	Value to write to the field

Details

This function returns *result*, which is *value* with a field of bits overwritten, starting at *index*. The *index* specifies the position of the least significant bit of *value*. The *width* bits starting at *index* are set to *fieldValue*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Before setting the field of bits, any fractional parts of *value* and *fieldValue* are truncated to form integers.

If *fieldValue* is wider than *width*, the most significant bits of the *fieldValue* that exceed the width are truncated. For example, if *width* is 4 bits and the binary value for *fieldValue* is 11110 (5 bits), the most significant bit of *fieldValue* is truncated and a binary value of 1110 is used.

Example

`testResult = bit.setfield(15, 2, 3, 5)` `print(testResult)`	The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at *index* position 2, the returned `value` is decimal 11 (binary 1011). Output: `1.10000e+01`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.get() (on page 8-13)
bit.set() (on page 8-15)
bit.getfield() (on page 8-14)

bit.test()

This function returns the Boolean value (`true` or `false`) of a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`result = bit.test(value, index)`

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to test (1 to 32)

Details

This function returns *result*, which is the result of the tested bit.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for *value* is 0, *result* is `false`. If the bit of *value* at *index* is 1, the returned value is `true`.

If *index* is bigger than the number of bits in *value*, the result is `false`.

Example

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

Also see

Bit manipulation and logic operations (on page 6-5)
bit.clear() (on page 8-13)
bit.get() (on page 8-13)
bit.set() (on page 8-15)
bit.toggle() (on page 8-17)

bit.toggle()

This function toggles the value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.toggle(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to toggle (1 to 32)

Details

This function returns *result*, which is the result of toggling the bit *index* in *value*.

Any fractional part of *value* is truncated to make it an integer. The returned value is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The indexed bit for *value* is toggled from 0 to 1, or 1 to 0.

Example

`testResult = bit.toggle(10, 3)` `print(testResult)`	The binary equivalent of decimal 10 is 1010. Toggling the bit at *index* position 3 returns a decimal value of 14 (binary 1110). Output: `1.40000e+01`

Also see

Bit manipulation and logic operations (on page 6-5)
bit.clear() (on page 8-13)
bit.get() (on page 8-13)
bit.set() (on page 8-15)
bit.test() (on page 8-16)

bufferVar.appendmode

This attribute sets the state of the reading buffer's append mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Saved setup	See **Details**

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

state	The reading buffer append mode; set to one of the following: • `0`: Append mode off; new measurement data overwrites the previous buffer content • `1`: Append mode on; appends new measurement data to the present buffer content
bufferVar	The reading buffer

Details

Assigning a value to this attribute enables or disables the buffer append mode. This value can only be changed with an empty buffer. Use *bufferVar*`.clear()` to empty the buffer.

When a buffer is created over a remote interface, the append mode attribute default setting is off (`0`). However, when using the front panel or web interface, the default setting is on (`1`) to allow triggered readings to fill a buffer without clearing the previous readings.

If the append mode is set to `0`, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to `1`, any stored readings remain in the buffer and new readings are added to the buffer after the stored readings.

With append mode on, the first new measurement is stored at `rb[`*n*`+1]`, where *n* is the number of readings stored in buffer `rb`.

Example

`buffer1.appendmode = 1`	Append new readings to contents of the reading buffer named `buffer1`.

Also see

bufferVar.clear() (on page 8-23)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.basetimefractional

When enabled by the bufferVar.collecttimestamps attribute, this attribute contains the fractional portion of the timestamp (in seconds) for the first reading stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

`value = bufferVar.basetimefractional`

value	The fractional seconds of the timestamp
bufferVar	The reading buffer

Details

The `bufferVar.basetimefractional` information from a reading buffer is only available if the `bufferVar.collecttimestamps` attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (`bufferVar.clear()`).

A read-only attribute for each existing reading buffer in the instrument.

The attribute represents the fractional seconds of the timestamp when reading 1 was stored in the buffer

Example

`baseFractional = buffer1.basetimefractional`	Read the `basetimefractional` attribute for `buffer1` and store it in a variable called `baseFractional`.

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.basetimeseconds

When enabled by the `bufferVar.collecttimestamps` attribute, this attribute represents the nonfractional seconds of the timestamp for the first reading stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

`value = bufferVar.basetimeseconds`

value	The nonfractional seconds of the timestamp
bufferVar	The reading buffer

Details

The basetime seconds information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

This attribute is a read-only attribute for each existing reading buffer in the instrument.

This attribute represents the nonfractional seconds of the timestamp when reading 1 was stored in the buffer.

Example

`basedSeconds = buffer1.basetimeseconds`	Read the `basetimeseconds` attribute for `buffer1` and store in a variable called `baseSeconds`.

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.cachemode

This attribute enables or disables the reading buffer cache (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Clearing the buffer cache Instrument reset Recall setup	Not saved	1 (enabled)

Usage

```
cacheMode = bufferVar.cachemode
bufferVar.cachemode = cacheMode
```

cacheMode	The reading buffer cache mode; set to one of the following: • 0: Cache mode disabled (off) • 1: Cache mode enabled (on)
bufferVar	The reading buffer

Details

Assigning a value to this attribute enables or disables the reading buffer cache. When enabled, the reading buffer cache improves access speed to reading buffer data.

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. To avoid this, make sure that you include commands that automatically invalidate the cache as needed (for example, explicit calls to the *bufferVar*.clearcache() function) or disable the cache using this attribute (*bufferVar*.cachemode).

Example

`buffer1.cachemode = 1`	Enables reading buffer cache.

Also see

bufferVar.clearcache() (on page 8-23)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.capacity

This attribute contains the capacity of the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup	Create configuration script Save setup	Not applicable

Usage

bufferCapacity = *bufferVar*.capacity

bufferCapacity	The maximum number of readings the buffer can store
bufferVar	The reading buffer

Details

This read-only attribute reads the number of readings that can be stored in the buffer.

Example

`bufferCapacity = buffer1.capacity` `print(bufferCapacity)`	Reads the capacity of a reading buffer named `buffer1`. Output: `1.00000e+05` The above output indicates that the buffer can hold 100000 readings.

Also see

Reading buffers (on page 3-55, on page 3-49)

bufferVar.channels

When enabled by the *bufferVar*.collectchannels attribute, this buffer recall attribute gets the channel, backplane relay, or channel pattern information stored with readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

channels = *bufferVar*.channels[*N*]

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The channels information from a reading buffer is only available if the *bufferVar*.collectchannels attribute is set to 1 (default setting). If it is set to 0, you will not be able to access the channels information from a reading buffer. You may change the collect channels setting when the buffer is empty (*bufferVar*.clear()).

This read-only attribute is an array (a Lua table) of strings indicating the channel or channel pattern associated with the measurement.

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

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

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collectchannels = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named testData, configures the buffer to collect channel data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.channels[1])`	The print() command then ouputs the first measurement channel. Output: 2035+
`printbuffer(1, 6, testData.channels)`	The printbuffer() command then ouputs the channels for measurements 1 to 6 in the reading buffer. Output: 2035+, 2036+, 2037+, 2038+, 2039+, 2040+

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collectchannels (on page 8-24)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.clear()

empties the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar.clear()

bufferVar	The reading buffer

Details

clears all readings and statistics from the specified buffer. (for example, time, status, channels, and units) from the specified buffer.

Example

`testData = dmm.makebuffer(50)` `testData.appendmode = 1` `dmm.measurecount = 3` `dmm.measure(testData)` `printbuffer(1,testData.n, testData )`	Create a reading buffer named `testData` and enable append mode for it. Take three readings and store them in `testData`, and then view the readings. Output: `3.515871341e-07, 5.596728126e-07, 3.944283032e-07`
	Next, clear the data and verify there are no readings in buffer.
`testData.clear()` `print("Readings in buffer after clear ="` `      .. testData.n)`	Output: `Readings in buffer after clear =  0`
`dmm.measurecount = 3` `dmm.measure(testData)` `printbuffer(1,testData.n, testData )`	Store three new readings in the buffer and view those when done. Output: `4.923509754e-07, 3.332266330e-07, 3.974883867e-07`

Also see

Reading buffers (on page 3-55, on page 3-49)

bufferVar.clearcache()

This function clears the cache.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar.clearcache()

bufferVar	The reading buffer

Details

This function clears all readings from the specified cache.

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when initiating successive scans without reconfiguring the scan measurements. Watch for this when running Lua code remotely on more than one node, because values in the reading buffer cache may change while the Lua code is running.To avoid this, you can include explicit calls to the `bufferVar.clearcache()` function to remove stale values from the reading buffer cache.

Example

`testData.clearcache()`	Clears the reading buffer cache for a user-defined buffer named `testData`.

Also see

bufferVar.cachemode (on page 8-20)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.collectchannels

This attribute sets the storage state of channel information with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	1 (enabled)

Usage

```
state = bufferVar.collectchannels
bufferVar.collectchannels = state
```

state	The reading buffer collect channels mode; set to one of the following: • `0`: Collect channels mode disabled (off); channel information is not stored in the reading buffer • `1`: Collect channels mode enabled (on); channel information is stored in the reading buffer
bufferVar	The reading buffer

Details

Assigning a value to this attribute enables or disables the storage of channel information, which includes channel, backplane relay, or channel pattern information associated with the reading. Reading this attribute returns the state of channel information collection.

When on, channel information is stored with readings in the buffer. This requires eight extra bytes of storage per reading.

This value, off (`0`) or on (`1`), can only be changed when the buffer is empty. Empty the buffer using the `bufferVar.clear()` function.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collectchannels = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect channel data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.channels[1])`	The `print()` command then ouputs the first measurement channel. Output: `2035+`
`printbuffer(1, 6, testData.channels)`	The `printbuffer()` command then ouputs the channels for measurements 1 to 6 in the reading buffer. Output: `2035+, 2036+, 2037+, 2038+, 2039+, 2040+`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.channels (on page 8-21)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.collecttimestamps

This attribute sets whether or not timestamp values are stored with the readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	1 (enabled)

Usage

```
state = bufferVar.collecttimestamps
bufferVar.collecttimestamps = state
```

state	Timestamp value collection status; set to one of the following: • `0`: Timestamp value collection disabled (off) • `1`: Timestamp value collection enabled (on)
bufferVar	The reading buffer

Details

Assigning a value to this attribute enables or disables the storage of timestamps. Reading this attribute returns the state of timestamp collection.

When on, timestamp values are stored with readings in the buffer. This requires four extra bytes of storage for each reading.

This value, off (0) or on (1), can only be changed when the buffer is empty. Empty the buffer using the *bufferVar*`.clear()` function.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect timestamp data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.timestamps[1])`	The `print()` command then outputs the first measurement timestamp. Output: `07/11/2011 09:14:48.509762161`
`printbuffer(1, 6, testData.timestamps)`	The `printbuffer()` command then outputs the timestamps for measurements 1 to 6 in the reading buffer. Output: `07/11/2011 09:14:48.509762161,` `07/11/2011 09:14:48.528708001,` `07/11/2011 09:14:48.547659196,` `07/11/2011 09:14:48.566612446,` `07/11/2011 09:14:48.585565606,` `07/11/2011 09:14:48.681325966`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.timestamps (on page 8-38)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.dates

When enabled by the *bufferVar*`.collecttimestamps` attribute, this attribute contains the dates (month, day, and year) of readings stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
dates = bufferVar.dates[N]
```

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The *bufferVar*`.dates` information from a reading buffer is only available if the *bufferVar*`.collecttimestamps` attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*`.clear()`).

This read-only attribute is an array (a Lua table) of strings indicating the date of the reading, formatted in month, day, and year format.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.dates[1])`	The `print()` command then outputs the first measurement date. Output: `07/11/2011`
`printbuffer(1, 6, testData.dates)`	The `printbuffer()` command then outputs the dates for measurements 1 to 6 in the reading buffer. Output: `07/11/2011, 07/11/2011,` `07/11/2011, 07/11/2011,` `07/11/2011, 07/11/2011`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.formattedreadings

This attribute contains the stored readings shown as they are formatted on the front-panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
readings = bufferVar.formattedreadings[N]
```

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

This attribute outputs an array (a Lua table) of strings that contain the stored readings. The readings are shown as they would appear on the front-panel display.

Use this attribute to access the reading elements *N* as they appear on the front panel.

Example

`reset()` `testData = dmm.makebuffer(1000)` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.formattedreadings[1])`	The `print()` command outputs the first reading, formatted as it appears on the front-panel display. Output: `+000.0006e-3`
`printbuffer(1, 6, testData.formattedreadings)`	The `printbuffer()` command then outputs readings 1 to 6 in the reading buffer as they appear on the front-panel display. Output: `6.000000000e-07, 7.000000000e-07,` `5.000000000e-07, 7.000000000e-07,` `7.000000000e-07, 6.000000000e-07`

Also see

bufferVar.readings (on page 8-31)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.fractionalseconds

This attribute contains the fractional portion of the timestamp (in seconds) when each reading occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
fractionalseconds = bufferVar.fractionalseconds[N]
```

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The *bufferVar*.fractionalseconds information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

This read-only attribute is an array (a Lua table) of the fractional portion of the timestamps, in seconds, of when each reading occurred. These are absolute fractional times.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.fractionalseconds[1])`	The `print()` command outputs the fractional portion of the timestamp for the first measurement in the buffer. Output: `5.097621610e-01`
`printbuffer(1, 6, testData.fractionalseconds)`	The `printbuffer()` command then outputs the fractional portion of the timestamp for the first six measurements in the buffer. Output: `5.097621610e-01, 5.287080010e-01,` `5.476591960e-01, 5.666124460e-01,` `5.855656060e-01, 6.813259660e-01`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.n

This attribute contains the number of readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer Reset Recall setup	Not saved	Not applicable

Usage

`numberOfReadings` `=` *`bufferVar`*`.n`

`numberOfReadings`	The number of readings stored in the buffer
`bufferVar`	The reading buffer

Details

This read-only attribute contains the number of readings presently stored in the buffer.

Example

`numberOfReadings = buffer1.n` `print(numberOfReadings)`	Reads the number of readings stored in a reading buffer named `buffer1`. Output: `1.250000+02` The above output indicates that there are 125 readings stored in the buffer.

Also see

bufferVar.formattedreadings (on page 8-27)
bufferVar.fractionalseconds (on page 8-28)
bufferVar.readings (on page 8-31)
bufferVar.relativetimestamps (on page 8-32)
bufferVar.seconds (on page 8-34)
bufferVar.statuses (on page 8-35)
bufferVar.times (on page 8-36)
bufferVar.units (on page 8-39)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.ptpseconds

When enabled by the *bufferVar*.collecttimestamps attribute, this attribute contains the absolute seconds portion of the timestamp of when the reading was stored, in PTP format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

`value = bufferVar.ptpseconds[N]`

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The ptp seconds information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

These seconds are absolute and in PTP format.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.ptpseconds[1])`	The `print()` command outputs the absolute seconds portion of the timestamp of first measurement in the buffer, in PTP format. Output: `1.310375688e+09`
`printbuffer(1, 6, testData.ptpseconds)`	The `printbuffer()` command outputs the absolute seconds portion of the timestamp for measurements 1 to 6 in the reading buffer, in PTP format. Output: `1.310375688e+09, 1.310375688e+09,` `1.310375688e+09, 1.310375688e+09,` `1.310375688e+09, 1.310375688e+09`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

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	Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

`reading` `=` *`bufferVar`*`.readings[`*`N`*`]`

`reading`	The value of the reading in the specified reading buffer
`bufferVar`	The reading buffer
`N`	The reading number (1 to *`bufferVar`*`.n`)

Details

The `readings` buffer recall attribute is like an array (a Lua table) of the readings stored in the reading buffer. This array holds the same data that is returned when the reading buffer is accessed directly; that is, `rb[2]` and `rb.readings[2]` access the same value.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collectchannels = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect channel data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.readings[1])`	The `print()` command then outputs the first reading in the reading buffer. Output: `6.239269805e-07`
`printbuffer(1, 6, testData.readings)`	The `printbuffer()` command then outputs the readings for measurements 1 to 6 in the reading buffer. Output: `6.239269805e-07, 6.943093615e-07, 4.954026325e-07, 7.432710179e-07, 6.943093615e-07, 6.331072911e-07` NOTE: An alternative way to use the `printbuffer()` command for this example is `printbuffer(1, 6, testData)`, because `"readings"` is an optional parameter and is assumed if it has not been specified.

Also see

bufferVar.formattedreadings (on page 8-27)
bufferVar.fractionalseconds (on page 8-28)
bufferVar.relativetimestamps (on page 8-32)
bufferVar.seconds (on page 8-34)
bufferVar.statuses (on page 8-35)
bufferVar.times (on page 8-36)
bufferVar.units (on page 8-39)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.relativetimestamps

When enabled by the *bufferVar*`.collecttimestamps` attribute, this attribute contains the timestamps, in seconds, of when each reading occurred relative to the timestamp of reading buffer entry number 1.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	1

Usage

```
relativetimestamp = bufferVar.relativetimestamps[N]
```

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The relative timestamps information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

This read-only attribute is an array (a Lua table) of timestamps, in seconds, of when each reading occurred relative to the timestamp of reading buffer entry number 1. These timestamps are equal to the time that has lapsed for each reading since the first reading was stored in the buffer. Therefore, the relative timestamp for entry number 1 in the buffer will equal 0.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.relativetimestamps[1])`	The `print()` command then outputs the relative timestamp of the first measurement in the reading buffer. Output: `0.000000000e+00`
`printbuffer(1, 6, testData.relativetimestamps)`	The `printbuffer()` command then outputs the relative timestamp for measurements 1 to 6 in the reading buffer. Output: `0.000000000e+00,` `1.894584000e-02,` `3.789703500e-02,` `5.685028500e-02,` `7.580344500e-02,` `1.715638050e-01`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.seconds

When enabled by the *bufferVar*.timestamps attribute, this attribute contains the nonfractional seconds portion of the timestamp when the reading was stored, in UTC format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
seconds = bufferVar.seconds[N]
```

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

The *bufferVar*.seconds information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you will not be able to access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

This read-only attribute is an array (a Lua table) of the seconds portion of the timestamp when the reading was stored, in seconds. These seconds are absolute and in UTC format.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.seconds[1])`	The `print()` command then ouputs the seconds portion of the timestamp of the first reading in the reading buffer. Output: `1.310375688e+09`
`printbuffer(1, 6, testData.seconds)`	The `printbuffer()` command then ouputs the seconds portion of the timestamps for measurements 1 to 6 in the reading buffer. Output: `1.310375688e+09, 1.310375688e+09,` `1.310375688e+09, 1.310375688e+09,` `1.310375688e+09, 1.310375688e+09`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

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	Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

statusInformation	The status value when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

This read-only buffer recall attribute is like an array (a Lua table) of the status values for all the readings in the buffer. The status values are floating-point numbers that encode the status value; see the following table for values.

Buffer status bits

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

Example

```
reset()
testData = dmm.makebuffer(1000)
testData.collectchannels = 1
dmm.nplc = 0.5
dmm.range = 0
dmm.configure.set("Dcv_100mV")
dmm.setconfig("slot2", "Dcv_100mV")
scan.create("2035:2040")
scan.execute(testData)

print(testData.statuses[1])

printbuffer(1, 6, testData.statuses)
```

This example creates a reading buffer named `testData`, configures the buffer to collect channel data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
The `print()` command then ouputs the status value of the first measurement channel in the reading buffer.
Output:
`0.000000000e+00`

The `printbuffer()` command then ouputs the status values for measurements 1 to 6 in the reading buffer.
Output:
`0.000000000e+00, 0.000000000e+00,`
`0.000000000e+00, 0.000000000e+00,`
`0.000000000e+00, 0.000000000e+00`

Also see

bufferVar.timestamps (on page 8-38)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.times

When enabled by the *`bufferVar`*`.collecttimestamps` attribute, this attribute contains the time of the readings (in hours, minutes, and seconds format) in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

```
time = bufferVar.times[N]
```

`bufferVar`	The reading buffer
`N`	The reading number (1 to *`bufferVar`*`.n`)

Details

The times information from a reading buffer is only available if the *bufferVar*.collecttimestamps attribute is set to 1 (default setting). If it is set to 0, you cannot access any time information from a reading buffer. You may change the collect timestamps setting when the buffer is empty (*bufferVar*.clear()).

This read-only attribute is an array (a Lua table) of strings indicating the time of the reading formatted in hours, minutes, and seconds.

These seconds are absolute and in UTC format.

Example

`reset()` `testData = dmm.makebuffer(1000)` `testData.collecttimestamps = 1` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, configures the buffer to collect time and date data, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.times[1])`	The `print()` command then ouputs the time of the first reading in the reading buffer. Output: `09:14:48`
`printbuffer(1, 6, testData.times)`	The `printbuffer()` command then ouputs the time of readings 1 to 6 in the reading buffer. Output: `09:14:48, 09:14:48, 09:14:48,` `09:14:48, 09:14:48, 09:14:48`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.timestampresolution

This attribute contains the timestamp's resolution.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Reset Recall setup	Not saved	Not applicable

Usage

resolution = *bufferVar*.timestampresolution

resolution	Timestamp resolution in seconds
bufferVar	The reading buffer

Details

Reading this attribute returns the timestamp resolution value.

The finest timestamp resolution is 0.000001 seconds (1 μs). At this resolution, the reading buffer can store unique timestamps for up to 71 minutes. This value can be increased for very long tests.

The value specified when setting this attribute will be rounded to an even power of 2 μs.

Example

`buffer1.timestampresolution = 0.000008`	Sets the timestamp resolution of reading buffer 1 to 8 μs.

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
bufferVar.timestamps (on page 8-38)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.timestamps

When enabled by the *bufferVar*`.collecttimestamps` attribute, this attribute contains the timestamp (in seconds) of when each reading saved in the specified reading buffer occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Buffer storage settings Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

timestamp = *bufferVar*`.timestamps[`*N*`]`

timestamp	The timestamp of reading number *N* in the specified buffer when the reading was acquired
bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*`.n`)

Details

The *bufferVar*`.timestamps` information from a reading buffer is only available if the *bufferVar*`.collecttimestamps` attribute is set to 1 (default setting). If it is set to 0, you cannot access any time information from a reading buffer.

If enabled, this buffer recall attribute is like an array (a Lua table) that contains timestamps, in seconds, of when each reading occurred. These are relative to the *bufferVar*`.basetimestamp` for the buffer. See Reading buffer commands for more information.

Example

`printbuffer(1, 6, buffer1.timestamps)`	Print the timestamp of the first 6 readings stored in buffer 1. Example output: `07/11/2011 09:14:48.509762161,` `07/11/2011 09:14:48.528708001,` `07/11/2011 09:14:48.547659196,` `07/11/2011 09:14:48.566612446,` `07/11/2011 09:14:48.585565606,` `07/11/2011 09:14:48.681325966`

Also see

bufferVar.clear() (on page 8-23)
bufferVar.collecttimestamps (on page 8-25)
bufferVar.measurefunctions
bufferVar.measureranges
bufferVar.n (on page 8-29)
bufferVar.readings (on page 8-31)
bufferVar.sourcefunctions
bufferVar.sourceoutputstates
bufferVar.sourceranges
bufferVar.sourcevalues
bufferVar.statuses (on page 8-35)
Reading buffers (on page 3-55, on page 3-49)

bufferVar.units

This attribute contains the unit of measure stored with readings in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Clearing the buffer Instrument reset Recall setup	USB flash drive using dmm.savebuffer or dmm.appendbuffer	Not applicable

Usage

`units = bufferVar.units[N]`

bufferVar	The reading buffer
N	The reading number (1 to *bufferVar*.n)

Details

This attribute is an array (Lua table) of the strings that indicate the unit of measure that is stored with readings in the buffer. You can designate units as one of the following: Volts AC, Volts DC, Amps AC, Amps DC, dB VAC, dB VDC, Ohms 2wire, Ohms 4wire, Ohms ComSide, Fahrenheit, Kelvin, Celsius, Hertz, Seconds, and Continuity.

Example

`reset()` `testData = dmm.makebuffer(1000)` `dmm.nplc = 0.5` `dmm.range = 0` `dmm.configure.set("Dcv_100mV")` `dmm.setconfig("slot2", "Dcv_100mV")` `scan.create("2035:2040")` `scan.execute(testData)`	This example creates a reading buffer named `testData`, sets and saves the DMM configuration, creates a scan list, and then runs the scan.
`print(testData.units[1])`	The `print()` command outputs the units of the first reading in the reading buffer. Output: `Volts DC`
`printbuffer(1, 6, testData.units)`	The `printbuffer()` command outputs the units of readings 1 to 6 in the reading buffer. Output: `Volts DC, Volts DC, Volts DC,` `Volts DC, Volts DC, Volts DC`

Also see

Reading buffers (on page 3-55, on page 3-49)

channel.calibration.adjustcount()

This function gets the number of times that a card has been adjusted.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`count = channel.calibration.adjustcount(channelList)`

count	Return value representing the number of times the instrument has been adjusted
channelList	A string contains slot*X*, where *X* is a number from 1 to 6

Details

This command can be used with channels that are locked or unlocked. If no *channelList* is provided, the currently unlocked channels are assumed.

There is only one adjustment count per card. Therefore, with no channel unlocked, the only acceptable values for *channelList* are "slot1", "slot2", and so on. An error is generated if any other values are used.

Example

```
Count = channel.calibration.adjustcount("slot1")
print(Count)
```

Assign the number of times the card in slot 1 has been adjusted to a user variable named Count.
Output the value.
3
This shows that the instrument has been adjusted 3 times.

Also see

Channel list notation
channel.calibration.adjustdate() (on page 8-42)

channel.calibration.adjustdate()

This function sets or gets the adjustment date in UTC format (number of seconds since January 1, 1970) on the unlocked channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
returnDate = channel.calibration.adjustdate(channelList)
returnDate = channel.calibration.adjustdate(channelList, date)
```

returnDate	The adjustment date in UTC format
channelList	A string contains slot*X*, where *X* is a number from 1 to 6
date	The date of the adjustment (UTC formatted)

Details

This command can get the adjust date whether calibration is currently locked or unlocked. If the *channelList* parameter is not specified, it uses the currently unlocked card.

This command can only set the adjustment date on a previously unlocked card. The date is not permanently saved until the `channel.calibration.save()` command is sent.

There is only one adjustment date per card. Therefore, the only acceptable values for *channelList* are slot*X.* An error is generated if any other values are used.

Example 1

`adjustmentDate = channel.calibration.adjustdate("slot2")`	Gets the adjustment date for the card in slot 2.

Example 2

```
NewAdjustDate = os.time{year=2010, month=12, day=28, hour=17, min=35, sec=0}
channel.calibration.unlock("slot5", "KI3706")
myDate = channel.calibration.adjustdate("slot5", NewAdjustDate)
channel.calibration.save()
channel.calibration.lock()
print(os.date("%c", myDate))
```

Assign the UTC time for December 28, 2010 at 17:35:00 GMT to NewAdjustDate.
Unlock the calibration for the card in slot 5, assuming the default password.
Set the adjustment date using NewAdjustDate for the card in slot 5.
Save the adjustment date on the card on slot 5.
Lock the calibration for the card in slot 5.
View the date for myDate.

Also see

ChannelList notation
Lua date and time (see Lua date and time - *http://www.lua.org/pil/22.1.html*)
UTC Calculator (see UTC Calculator - *http://www.mbari.org/staff/rich/utccalc.htm*)
channel.calibration.adjustcount() (on page 8-41)
channel.calibration.save() (on page 8-45)
channel.calibration.verifydate() (on page 8-48)

channel.calibration.lock()

This function prevents further calibration on the currently unlocked card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.calibration.lock()
```

Details

Calibration data is locked during normal operation. To perform calibration, calibration must be unlocked (`channel.calibration.unlock()`) for the card.

Only one card can be calibrated at a time. Therefore, `channel.calibration.lock()` works only on the currently unlocked card. Once locked, you must unlock calibration to perform it again.

This command locks calibration on the card being calibrated, but does not save calibration data.

CAUTION

Calibration data is lost if it is not saved before locking. Refer to `channel.calibration.save()` for more information.

An error is generated if this command is issued when calibration is already locked.

Example

```
channel.calibration.unlock("slot1","KI3706")
-- Perform operations to generate the calibration data
channel.calibration.save()
channel.calibration.lock()
```

Unlock the card calibration for slot 1 using the default password.
Use the `channel.calibration.step` command to generate the calibration data.
Save the calibration data for the card in slot 1, if no errors occurred while generating the calibration data.
Lock the calibration data for the card in slot 1.

Also see

channel.calibration.save() (on page 8-45)
channel.calibration.step() (on page 8-46)
channel.calibration.unlock() (on page 8-47)

channel.calibration.password()

This function sets the password needed to unlock the calibration functionality of a card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No	Not applicable	Card nonvolatile memory	KI3706

Usage

```
channel.calibration.password(password)
```

password	A string of characters that contain the desired password

Details

There is only one password per card. Therefore, `channel.calibration.password()` works only on the currently unlocked card.

Make note of the `password`, because there is no command to query for the password once it has been set on the instrument. The password is not permanently saved until the `channel.calibration.save()` command is sent. Passwords are alphanumeric and case-sensitive.

This command generates an error if calibration is locked or if the password string length is greater than six characters.

The default password from the factory is KI3706. The first two characters in the password are capital K capital I (for Keithley Instruments).

Example

`channel.calibration.unlock("slot3","KI3706")`	Unlock the calibration for the card in slot 3, assuming the default password is still valid.
`channel.calibration.password("Unlock")`	Set the password to "Unlock" for the card in slot 3.
`channel.calibration.save()`	Saved the password for the card in slot 3 for subsequent unlocks.
`channel.calibration.lock()`	Lock the calibration for the card in slot 3.

Also see

channel.calibration.lock() (on page 8-43)
channel.calibration.unlock() (on page 8-47)

channel.calibration.save()

This function saves the calibration data to the card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.calibration.save()
```

Details

Only one card can be calibrated at a time. Therefore, `channel.calibration.save()` works only on the presently unlocked card. An error is generated if this command is issued when calibration is already locked.

The system must receive this command before the `channel.calibration.lock()` command or the calibration data will be lost.

This command saves the present values of the calibration constants and calibration date, and increases the calibration count by one, regardless of errors in the data. You should not issue `channel.calibration.save()` unless the calibration procedure was performed with no errors.

If no calibration date was specified using either `channel.calibration.adjustdate()` or `channel.calibration.verifydate()`, the date is automatically assigned based on the system date.

Example

```
channel.calibration.unlock("slot1","KI3706")
-- Perform operations to generate the calibration data
channel.calibration.save()
channel.calibration.lock()
```

Unlock the card calibration for slot 1 using the default password.
Use the `channel.calibration.step` command to generate the calibration data.
Save the calibration data for the card in slot 1, if no errors occurred while generating the calibration data.
Lock the calibration data for the card in slot 1.

Also see

channel.calibration.adjustcount() (on page 8-41)
channel.calibration.adjustdate() (on page 8-42)
channel.calibration.lock() (on page 8-43)
channel.calibration.unlock() (on page 8-47)
channel.calibration.verifydate() (on page 8-48)

channel.calibration.step()

This function sends a calibration command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.calibration.step(channel, step)
channel.calibration.step(channel, step, value)
```

channel	The channel to be calibrated
step	The number corresponding to the specified step
value	The measured value for the specified step when the step value is even

Details

The specified channel must be on the unlocked slot. Only DAC and totalizer channels can be calibrated. It is best to calibrate a single channel sequentially to completion before changing channels.

The card assumes that the given voltage or current value is exactly what it is sourcing for the given step. This command generates an error if the step is out of sequence, does not exist, or the calibration is locked. Also, an error is generated if the calibration step does not complete successfully, if the value passed is invalid or out of range for the step, or not needed.

For DAC channels, a calibration sequence includes these steps:

1. Set voltage, –12 V to +12 V range, generate negative point 1.
2. Send reading.
3. Set voltage, –12 V to +12 V range, generate negative point 2.
4. Send reading.
5. Set voltage, –12 V to +12 V range, generate positive point 1.
6. Send reading.
7. Set voltage, –12 V to +12 V range, generate positive point 2.
8. Send reading.
9. Set current, 0 mA to +20 mA range, generate point 1.
10. Send reading.
11. Set current, 0 mA to +20 mA range, generate point 2.
12. Send reading.
13. Set current, +4 mA to +20 mA range, generate point 1.
14. Send reading.
15. Set current, +4 mA to +20 mA range, generate point 2.
16. Send reading.

For totalizer channels, a calibration sequence includes these steps:

1. Calibrate 0 V totalizer threshold
2. Calibrate 1.5 V totalizer threshold

You must save the calibration after calibrating and before locking. Use `channel.calibration.save()` to save the calibration.

NOTE

All calibration progress is lost if the calibration data is not saved before you lock the channel.

After calibration, the channel must be locked using `channel.calibration.lock()`.

Also see

ChannelList notation
channel.calibration.lock() (on page 8-43)
channel.calibration.save() (on page 8-45)
channel.calibration.unlock() (on page 8-47)

channel.calibration.unlock()

This function unlocks calibration functionality for a card so that calibration operations can be performed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`channel.calibration.unlock(`*`slotX, password`*`)`

slotX	A string containing *slotX*, where X is a slot number from 1 to 6
password	The password that unlocks calibration

Details

Calibration data is locked during normal operation. This command enables calibration functionality. When calibration is completed, calibration functionality must once again be locked (`channel.calibration.lock()`). Only one card at a time may be unlocked.

There is only one password per card. Therefore, the only acceptable values for channel list are `"slot1"`, `"slot2"`, and so on. Otherwise, an error is generated.

An error is generated if the password that is entered does not match the one that was saved with `channel.calibration.password()`.

The password can only contain six case-sensitive, alphanumeric characters.

The default password from the factory is **KI3706**. The first two characters in the password are capital K capital I (for Keithley Instruments).

Example

```
channel.calibration.unlock("slot1","KI3706")
-- Perform operations to generate the calibration data
channel.calibration.save()
channel.calibration.lock()
```

Also see

channel.calibration.lock() (on page 8-43)
channel.calibration.password() (on page 8-44)
channel.calibration.save() (on page 8-45)

channel.calibration.verifydate()

This function gets or sets the date the calibration was verified in UTC format (number of seconds since January 1, 1970).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
returnDate = channel.calibration.verifydate(slotX)
returnDate = channel.calibration.verifydate(slotX, date)
```

returnDate	The verification date returned from the function call
slotX	A string containing `slotX`, where *X* is 1 to 6
date	UTC formatted date to which to set the calibration verification date

Details

This command gets the verification date whether calibration is currently locked or unlocked. If the slot is not defined, the unlocked channel is assumed.

This command can only set the verification date on a previously unlocked card. The date is not permanently saved until `channel.calibration.save()` is issued.

There is only one verification date per card. If more than one slot is defined, an error is generated.

Example

```
channel.calibration.unlock("slot1", "KI3706")
print(channel.calibration.verifydate(os.time{year=2010, month=8, day=5}))
channel.calibration.save()
channel.calibration.lock()
print(os.date("%m/%d/%Y", channel.calibration.verifydate("slot1")))
```

Unlock the calibration for the card in slot 1 using the default password.
Set the verify calibration date to August 5, 2010.
Get the newly set verification date in a user-readable format.
Save the new verification date.
Lock the calibration.
Output:

```
1281009600
08/05/2010
```

Also see

Lua date and time (see Lua date and time - *http://www.lua.org/pil/22.1.html*)
UTC calculator (see UTC Calculator - *http://www.mbari.org/staff/rich/utccalc.htm*)
channel.calibration.adjustdate() (on page 8-42)
channel.calibration.save() (on page 8-45)

channel.clearforbidden()

This function clears the list of channels specified from being forbidden to close.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.clearforbidden(channelList)
```

channelList	String that specifies a list of channels, using channel list notation

Details

The *channelList* parameter indicates the channels that will no longer be forbidden to close, and may include:

- `allslots` or `slotX` (where *X* equals 1 to 6)
- Channel ranges or individual channels
- Analog backplane relays

This function allows all items contained in the *channelList* parameter to be closed. It removes the "forbidden to close" attribute that can be applied to a channel using `channel.setforbidden()`.

Command processing stops as soon as an error is detected. If an error is found, the channels are not cleared from being forbidden to close.

Example

`channel.clearforbidden("2002,2004,2006,2008")`	Clears channels 2, 4, 6, and 8 on slot 2 from being forbidden to close.
`channel.clearforbidden("allslots")`	Clears all channels from being forbidden to close.
`channel.clearforbidden("3005:3010")`	Clears channels 5 through 10 on slot 3 from being forbidden to close.

Also see

channel.getforbidden() (on page 8-66)
channel.setforbidden() (on page 8-94)

channel.close()

This function closes the channels, analog backplane relays, and channel patterns that are specified by the channel list parameter without opening any channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.close(channelList)`

channelList	A string that lists the channels, analog backplane relays, and channel patterns to close

Details

Channels closed with this command are appended to the already closed channels (no previously closed channels are opened by this command).

The *channelList* parameter can include channels with analog backplane relays. If this is the case, `channel.close` closes the specified channels and any associated analog backplane relays. For channel patterns, the analog backplane relays that are closed are the ones that were specified when the pattern was created. However, for channels, they are the ones specified with the `channel.setbackplane()` function. Another option for closing analog backplane relays with this command is to include them in the *channelList* parameter.

This command has no effect on how the DMM is configured.

Actions associated with this function include:

- Close the specified items in *channelList*
- Incur the settling time and any user-specified delay

This command is not available for digital I/O, DAC, and totalizer channels. Calling on a specific channel generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, the channel is ignored.

- For delay time, see `channel.setdelay()`
- For analog backplane relays with channels, see `channel.setbackplane()`
- For channels associated with a channel, see `channel.getimage()`
- For channels associated with a channel pattern, see `channel.pattern.getimage()`
- For channel states (open/close), see `channel.getstate()`
- For closed channels, see `channel.getclose()`.

An error is generated if:

- The parameter string contains `slotX`, where *X* is 1 to 6, or `allslots`
- A forbidden item is specified
- Specified channel does not support being closed, such as a digital I/O channel
- Channel is paired with another bank for a multiwire application

Once an error is detected, the command stops processing and no channels are closed.

Example

`channel.close("1001:1005, 3003, Chans")`	Close channels 1 to 5 on slot 1, channel 3 on slot 3, and the channel pattern or label `Chans`.
`channel.close("2001, 2913")`	Close channel 1 on slot 2 and analog backplane relay 3 in bank 1 on slot 2.

Also see

channel.exclusiveclose() (on page 8-56)
channel.exclusiveslotclose() (on page 8-57)
channel.getclose() (on page 8-61)
channel.open() (on page 8-79)
channel.getimage() (on page 8-67)
channel.getstate() (on page 8-75)
channel.pattern.getimage() (on page 8-81)
channel.pattern.snapshot() (on page 8-84)
channel.pattern.setimage() (on page 8-82)
channel.setbackplane() (on page 8-90)
channel.setdelay() (on page 8-93)
dmm.close() (on page 8-167)

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.BREAK_BEFORE_MAKE

Usage

```
rule = channel.connectrule
channel.connectrule = rule
```

rule	• `channel.BREAK_BEFORE_MAKE` or `1`: Break-before-make (BBM) connections for relays in the instrument • `channel.MAKE_BEFORE_BREAK` or `2`: Make-before-break (MBB) connections for relays in the instrument • `channel.OFF` 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 when using `channel.exclusiveclose()`, `channel.exclusiveslotclose()`, `dmm.close()`, and scanning commands like `scan.execute()` and `scan.background()`. These commands may both open and close switch channels in a single command. The connection rule dictates the algorithm used by the instrument to order the opening and closing of switches.

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 `channel.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 `channel.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.OFF`), 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 set to OFF. It is highly recommended that you implement cold switching when the channel connect rule is set to OFF.

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:

- `channel.BREAK_BEFORE_MAKE` open settle time + close settle time
- `channel.MAKE_BEFORE_BREAK` close settle time + open settle time

- `channel.OFF` maximum of open settle time or close settle time

This behavior is also affected by `channel.connectsequential` and any additional user delay times.

WARNING

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.connectsequential (on page 8-53)
channel.exclusiveclose() (on page 8-56)
channel.exclusiveslotclose() (on page 8-57)
dmm.close() (on page 8-167)
scan.background() (on page 8-323)
scan.execute() (on page 8-327)

channel.connectsequential

This attribute controls whether or not channels are closed sequentially.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script	channel.OFF

Usage

```
sequential = channel.connectsequential
channel.connectsequential = sequential
```

`sequential`	• `channel.OFF` or `0`: Disable sequential connections • `channel.ON` or `1`: Enable sequential connections

Details

When `channel.connectsequential` is enabled, the list of channel actions is acted on sequentially. No two relays are opened or closed simultaneously.

Using a sequential close allows you to determine the time for a close operation to happen. For example, if you close three channels and each takes 4 ms to closej (assuming no additional user delay times), with sequential on, it will take 12 ms. With sequential off, it may be 4, 8 or 12 ms, depending on whether or not the card can close multiple channels at once.

The order in which channels are opened or closed is not guaranteed with sequential off.

The sequential setting affects all channels in the instrument.

Example

`channel.connectsequential = channel.ON`	Specifies that channels close sequentially.

Also see

channel.connectrule (on page 8-52)
Switch operation (on page 2-77)

channel.createspecifier()

This function creates a string channel descriptor from a series of card-dependent integer arguments.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

There are five variants of this function that can be used, depending upon the type of card in the specified slot:

Variant 1	<string> = channel.createspecifier(<slot>, <bank>, <row>, <column>)
Variant 2	<string> = channel.createspecifier(<slot>, <row>, <column>)
Variant 3	<string> = channel.createspecifier(<slot>, <bank>, <index>)
Variant 4	<string> = channel.createspecifier(<slot>, <index>)
Variant 5	<string> = channel.createspecifier(<slot>, <backplane>)

slot	Specifies the slot number to use
bank	Specifies the bank number to use (if applicable)
row	Specifies the row number to use
column	Specifies the column number to use
index	Specifies the index to use
backplane	Specifies the backplane to use

Details

The arguments are dependent upon the card type in the specified slot. This command can only create valid channel descriptors; if an illegal argument is sent for the type of card in the specified slot, an error is generated.

Variants of this function can be used, depending on the type of card in the specified slot:

Type of card in slot	Code variants to use
Matrix card containing banks	Variant 1 or 5
Matrix card without banks	Variant 2 or 5
Multiplexer cards	Variant 3, 4, or 5

Example 1

```
cd = channel.createspecifier(3, 1, 2, 101)
print(cd)
```

Creates a channel descriptor on the Model 3732 card configured as a single 4x112 matrix in slot 3, bank 1, row 2, column 101.

Output:

`312A1`

Example 2

```
for row = 1,8 do
   for col = 1,28 do
      ch = channel.createspecifier(1,1,row,col)
      channel.setpole(ch, 2)
   end
end
```

Sets the pole setting to 2 for all channels in bank 1 on a Model 3732 card configured as a dual 8x28 matrix in slot 1.

Example 3

`cd = createspecifier(2, 2, 1)` `print(cd)`	Creates a channel descriptor on the Model 3724 multiplexer card in slot 2, bank 2, index 1. Output: `2031`

Example 4

`cd = createspecifier(1, 911)` `print(cd)`	Creates a channel descriptor on the Model 3724 multiplexer card in slot 1, backplane 911. Output: `1911`

channel.exclusiveclose()

This function closes the specified channels so that they are the only channels that are closed on the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.exclusiveclose(channelList)
```

channelList	A string listing the channels (including analog backplane relays) and channel patterns to exclusively close

Details

This command allows you to close specific channels and open any other channels on the instrument. When you send this command, any presently closed channel opens if it is not specified to be closed in the parameter. For channel patterns, the analog backplane relays that are closed or opened are the ones that were specified when the pattern was created with `channel.pattern.setimage()` or `channel.pattern.snapshot()`. For channels, the analog backplane relays that are closed or opened are the ones specified with `channel.setbackplane()`, or that are specified in *channelList*.

When you send this command:

- Any presently closed channels and analog backplane relays that are not specified in *channelList* are opened.
- The channels and analog backplane relays in *channelList* are closed.
- Settling and user-specified delay times are applied as defined by the connection rules and delay settings.

This function has no affect on how the DMM is configured and does not use analog backplane relays associated with DMM configuration.

If the *channelList* parameter is an empty string or a string of spaces, all channels and analog backplane relays are opened. Therefore, sending `channel.exclusiveclose("")` is equivalent to `channel.open(channel.getclose("allslots"))`. However, sending the equivalent commands when nothing is closed generates an error because `nil` (the response of `channel.getclose("allslots")`) is being sent to the open command.

An error is generated if:

- The parameter string contains `slotX`, where X = 1 to 6 or `allslots`
- A specified channel or channel pattern is invalid
- Channel number does not exist for slot specified
- Slot is empty
- A forbidden item is specified
- Channel is paired with another bank for a multi-wire application

Once an error is detected, the command stops processing. Channels open or close only if no errors are found.

This command is not available for digital I/O, DAC, and totalizer channels. Calling on a specific channel for these channels generates an error. If the digital I/O, DAC, or totalizer channel is in the range of specified channels, the channel is ignored.

Example 1

`channel.setbackplane("3003","3913")` `channel.exclusiveclose("3003")`	Associate analog backplane relay 3 in bank 1 on slot 3 with channel 3 on slot 3. Open all channels and close slot 3, channel 3 and its associated analog backplane relay (3 in bank 1 on slot 3), if it is not already closed.

Example 2

`channel.exclusiveclose("3003, 3913")`	Close channel 3 on slot 3 and its associated analog backplane relay 3 in bank 1 on slot 3. By specifying the backplane relay directly, you eliminate the need for associating the backplane with `channel.setbackplane`.

Also see

channel.close() (on page 8-50)
channel.connectrule (on page 8-52)
channel.connectsequential (on page 8-53)
channel.exclusiveslotclose() (on page 8-57)
channel.getclose() (on page 8-61)
channel.getimage() (on page 8-67)
channel.getstate() (on page 8-75)
channel.open() (on page 8-79)
channel.pattern.getimage() (on page 8-81)
channel.setbackplane() (on page 8-90)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84)
channel.setdelay() (on page 8-93)
dmm.close() (on page 8-167)

channel.exclusiveslotclose()

This function closes the specified channels and channel patterns on the associated slots and opens any channels that are not specified.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.exclusiveslotclose(channelList)`

`channelList`	A string that lists the channels and channel patterns to exclusively close on the cards in associated slots (you can specify analog backplane relays)

Details

This command allows you to bundle the closing of channels with the opening of channels. Any currently closed channels or analog backplane relays open if they are not specified to be closed on the slots related to the channels in *`channelList`*. Using this command guarantees that only the specified channels and channel patterns are closed on the slots associated with channels in the *`channelList`*.

For channel patterns, the analog backplane relays that are closed or opened are the ones that were specified when the pattern was created (see `channel.pattern.setimage()` or `channel.pattern.snapshot()`). For channels, the analog backplane relays are the ones specified with the `channel.setbackplane()` command. If you do not want to use the `channel.setbackplane()` command, you can close the analog backplane relays by including them in the *`channelList`* parameter.When this command is sent:

- Closed channels or analog backplane relays for the associated slots are opened if they are not specified in the *`channelList`*
- Channels or analog backplane relays specified by the items in *`channelList`* are closed
- Any settling times and user-specified delay times are incurred before command processing is complete

This function has no effect on how the DMM is configured

For example, if channel 1 is closed on each of the six slots, specifying a `channelList` parameter of `"2002, 4004"` with this command opens channel 1 on slots 2 and 4 only. Then, channel 2 on slot 2 and channel 4 on slot 4 close. Channel 1 remains closed on slots 1, 3, 5, and 6.

The command is not available for digital I/O, DAC, and totalizer channels. Calling on one of these channels generates an error. If the digital I/O, DAC, or totalizer channel is in the range of channels, the channel is ignored.

An error is generated if:

- The parameter string contains slot*X* (where *X* = 1 to 6) or allslots
- The parameter string is empty or parameter string with just spaces
- A specified channel is invalid or does not exist for the slot
- Channel pattern does not exist or the image of the pattern is an empty channel list
- A forbidden item is specified
- Channel is paired with another bank for a multi-wire application

Once an error is detected, the command stops processing. Channels open or close only if no errors are found and remain unchanged with any parsing or syntax error.

Example

```
channel.exclusiveslotclose("3003")
```

Close channel 3 on slot 3 and open all other channels on slot 3 without affecting any other slot.

```
channel.exclusiveslotclose("1005, 2005")
```

Close channel 5 on slots 1 and 2 and open all other channels on slots 1 and 2 without affecting any other slots.

```
channel.pattern.setimage("5007, 5017, 5027, 5915," "RouteA")
channel.exclusiveslotclose("RouteA")
```

Create a channel pattern called RouteA that includes channels 7, 17, and 27 on slot 5. Analog backplane relay 5 in bank 1 on slot 5 is also in the pattern. Have only the RouteA channels close on slot 5 (channels 7, 17, and 27, and analog backplane relay 5 in bank 1 on slot 5.

Also see

Channel list notation
channel.close() (on page 8-50)
channel.connectrule (on page 8-52)
channel.connectsequential (on page 8-53)
channel.exclusiveclose() (on page 8-56)
channel.getclose() (on page 8-61)
channel.getimage() (on page 8-67)
channel.open() (on page 8-79)
channel.pattern.getimage() (on page 8-81)
channel.setbackplane() (on page 8-90)
channel.setdelay() (on page 8-93)

channel.getbackplane()

Returns a string that lists the analog backplane relays that are controlled when the specified channels are used with switching operations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Pole setting change	Create configuration script Save setup	None or nil

Usage

analogBusList = channel.getbackplane(*channelList*)

analogBusList	A string listing analog backplane relays associated with items in *channelList*.
channelList	A string listing the channels being queried.

Details

The response indicates the analog backplane relays that are used during processing of the command:

- `channel.close()`
- `channel.exclusiveclose()`
- `channel.open()`
- `scan.execute()` or `scan.background()` if the channel is configured for switching

The response will be changed by `channel.setbackplane()`, replacing the analog backplane relays with the new specified list.

The response will be cleared if `channel.setpole()` sets a new pole selection.

The analog backplane relays indicated by this response are not used or affected by:

- `dmm.close()` or `dmm.open()`
- `scan.execute()` or `scan.background()` if channel is configured for measuring

The parameter string can contain `"slot`*`X`*`"`, where *X* equals 1 to 6, or `"allslots"`.

An error is generated if:

- A specified channel does not exist for the card installed in a slot
- A channel pattern is specified in parameter list
- A specified channel does not have analog backplane relays associated with it, such as digital I/O
- An analog backplane relay is specified in parameter list

When *channelList* contains multiple items, the string returned includes the analog backplane relay channels of a single channel separated by a comma. A semicolon is used to delineate channels.

For channel patterns, the analog backplane relays must be specified when creating the pattern in the channel list parameter — see `channel.pattern.setimage()` or `channel.pattern.snapshot()`. Therefore, to see the channels and analog backplane relays associated with a channel pattern, use the `channel.pattern.getimage()` function.

Command processing stops as soon as an error is detected and a `nil` response is then returned. No partial list is returned.

For digital I/O, DAC, and totalizer channels, nothing is returned.

Example

`channel.setpole("slot5", 4)` `channel.setbackplane("slot5", "5911, 5922")` `print(channel.getbackplane("slot5"))`	Assume a Model 3720 in slot 5. Set all channels on the card in slot 5 to be 4-pole, which makes the card have 30 4-pole channels. Set all channels in slot 5 to have associated analog backplane relays 911 and 922 on slot 5. Get the associated analog backplane relays for channels on slot 5. Output: `5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922,5911,5922`

Also see

Channel list notation
Data retrieval commands (on page 6-3)
channel.close() (on page 8-50)
channel.exclusiveclose() (on page 8-56)
channel.open() (on page 8-79)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84)
channel.setbackplane() (on page 8-90)
channel.setpole() (on page 8-101)
dmm.close() (on page 8-167)
dmm.open() (on page 8-219)
scan.background() (on page 8-323)
scan.execute() (on page 8-327)

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(channelList)
```

closed	A string listing the channels that are presently closed in the specified channel list parameter
channelList	A string representing the channels, channel patterns, and backplane relays that will be queried

Details

If more than one channel is closed, they are comma-delimited in the string. If `channelList` equals "`slot`*X*" (where *X* is 1 to 6), the response indicates the channels and backplane relays that are closed on that slot. Similarly, if *channelList* equals `"allslots"`, the response indicates all channels and analog backplane relays that are closed in the instrument. The format of each channel returned is slot, row, column (matrix channels) or slot, channel (MUX channels). When the *channelList* contains a channel pattern, only the closed channels in that image are returned.

You can use `"allslots"` to query for all channels closed and not worry about an error if one of the slots is empty or does not support close channels.

An error message is generated if an empty parameter string is specified or if the specified channel list contains no valid channels that can be closed (for example, a channel list equaling "`slot`*X*" or "`allslots`").

If nothing is closed within the specified scope, a `nil` response is returned.

Example 1

`channel.setpole("slot5", 4)`	Configure the channel on slot 5 to be four-pole.
`channel.setbackplane("slot5", "5911, 5922")`	Associate the slot 5 channels with analog backplane relays 911 and 922 on slot 5.
`channel.close("5003, 5005")`	Close channels 3 and 5 on slot 5.
`closedSlot5 = channel.getclose("slot5")` `print(closedSlot5)`	Gets the channels and analog backplane relays that are closed on slot 5 and output the closed channels on slot 5. Output: `5003(5033);5055(5035);5911;5922`

Example 2

`allClosed = channel.getclose("allslots")`	Gets all channels and analog backplane relays that are closed in the instrument.

Example 3

`closedChans = channel.getclose("Chans")`	Gets all channels closed in a pattern called `"Chans"`.

Example 4

`closedRange = channel.getclose("3001:3020")`	Gets all channels that are closed on channels 1 to 20 on slot 3.

Example 5

`closedOnes = channel.getclose("3001, 3002, 3003, 3005, 3911, 3912")`	Gets all channels that are closed on channels 1, 2, 3, 5 and analog backplane relay 1 and 2 in bank 1 on slot 3.

Also see

Channel list notation
channel.close() (on page 8-50)
channel.exclusiveclose() (on page 8-56)
channel.getstate() (on page 8-75)
channel.open() (on page 8-79)
Data retrieval commands (on page 6-3)

channel.getcount()

This function returns a string with the close counts for the specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
counts = channel.getcount(channelList)
```

counts	A comma-delimited string listing the channel close counts
channelList	A string listing the items to query, which can include: • Channels • Backplane relays • Channel patterns (channels will be listed in the order in which they are listed in the pattern) • `slot`*X*, where *X* equals 1 to 6 • `allslots`

Details

A close count is the number of times a relay has been closed. The count values are returned in the order in which the channels are specified. The close counts for an analog backplane relay can be included in the *channelList* parameter.

If *channelList* includes a pattern, you can use `channel.pattern.getimage()` with the pattern name to see the channel order and the channels to which the close counts pertain.

When the *channelList* parameter for this function is `"slotX"`, the response first lists the channels starting from lowest to highest (from slot 1 to slot 6). Because each slot is processed completely before going to the next, all slot 1 channels and backplane relays are listed before slot 2 channels.

An error is generated if:

- A specified channel is invalid
- The channel does not have a count closure associated with it

If an error is detected, a `nil` value is returned. No partial list of close counts is returned.

NOTE

Pseudocards do not support counts, so count values are generated numbers, not actual count values, if a pseudocard is used.

Example 1

`counts = channel.getcount("2001:2005")` `print(counts)`	Gets the close counts for channels 1 to 5 on slot 2. Example output for channels 2001, 2002, 2003, 2004, and 2005: `672,495,547,479,518`

Example 2

`counts = channel.getcount("slot2")` `print(counts)`	Get the close counts for all channels and analog backplane relays on slot 2 assuming a 3721 card is installed in the slot. Sample output that shows the counts for channels 1 to 41, analog backplane relays 911 to 917, analog backplane relays 921 to 928: `672,495,547,479,518,459,522,599,452,451,464,427,426,428,426,425,428,424,424,425,5,3,3,3,4,3,3,5,3,33,33,33,33,33,33,32,32,32,32,32,119,3,56,0,0,0,0,0,0,14,68,0,0,0,0,16,0`

Example 3

`channel.pattern.setimage("2003, 2005, 2023, 2915", "Path")` `PathList = channel.pattern.getimage("Path")` `print(PathList)` `print(channel.getcount(PathList))` `print(channel.getcount("Path"))`	Create the a pattern called Path, then get the close counts for channels and analog backplane relays in channel pattern called `"Path"` Sample output: 2003, 2005,2023,2915 547,518,3,0 547,518,3,0

Also see

Channel list notation
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
Data retrieval commands (on page 6-3)

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	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0

Usage

```
delayTimes = channel.getdelay(channelList)
```

`delayTimes`	A comma-delimited string consisting of the delay times (in seconds) for channels specified in *`channelList`*
`channelList`	A string listing the channels to query for their delay times

Details

The *`channelList`* parameter may contain `slotX` (where X equals 1 to 6) or `allslots`.

A command, after closing the state of channels, 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. User delay is incurred
5. Command completes

The delay times are comma-delimited in the same order that the items were specified in the *`channelList`* parameter. A value of zero (0) indicates that no additional delay time is incurred before a close command completes.

An error message is generated for the following reasons:

- The specified channels do not support a delay time
- A channel pattern is specified

Command processing stops as soon as an error is detected and a `nil` response is generated.

> **NOTE**
>
> Pseudocards do not support user delays, so this value is always zero (0) if a pseudocard is used.

Example

Example 1

`delaytime = channel.getdelay("5001, 5003")` `print(delaytime)`	Query channels 1 and 3 on slot 5 for their delay times. Example output: `0.000e+00,0.000e+00`

Example 2

`patternChannels =` `   channel.pattern.get("chans")` `DelayPatternTimes =` `   channel.getdelay(patternChannels)`	Gets the delay of the channels in the `chans` channel pattern if `chans` does not contain backplane relays. If it does contain backplane relays, you will receive error code 1115, "Parameter error invalid channel type in channel list."

Also see

Channel list notation
channel.setdelay() (on page 8-93)
Data retrieval commands (on page 6-3)

channel.getforbidden()

This function returns a string listing the channels and analog backplane relays in the channel list that are forbidden to close.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Permitted to close

Usage

forbiddenList = channel.getforbidden(*channelList*)

forbiddenList	Comma-delimited string listing the channels and analog backplane relays in the channel list that are forbidden to close
channelList	A string listing the channels, backplane relays, and channel patterns that are to be checked to see if they are forbidden to close

Details

The *channelList* parameter indicates which channels to check, and may include:

- allslots or slot*X* (where *X* equals 1 to 6)
- Channel ranges or individual channels
- Analog backplane relays
- Channel patterns

If there are no channels in the scope of the *channelList* that are on the forbidden list, the string returned is empty or nil. The format of the channels in the response string is slot, channel for multiplexer channels or slot, row, column for matrix channels.

Example 1

`Forbidden = channel.getforbidden("allslots")`	Query for the channels and analog backplane relays that are forbidden to close in the instrument.

Example 2

```
channel.setforbidden("3003, 3005, 3925")
Forbidden =
   channel.getforbidden("slot3")
print(Forbidden)
```

Set channels 3 and 5 and analog backplane relay 5 in bank 2 to forbidden to close on slot 3.
Query for the channels and analog backplane relays that are forbidden to close on slot 3.
Sample output:
3003,3005,3925

Example 3

```
Forbidden =
   channel.getforbidden("1911:1916" ..
   ",2004,2008,2012")
```

Query for channels and analog backplane relays in a specified list. This list is only checking channels and analog backplane relays 1 to 6 on slot 1 and channels 4, 8 and 12 on slot 2 and returning the channels and analog backplane relays that are forbidden to close.

Also see

Channel list notation
channel.clearforbidden() (on page 8-49)
channel.setforbidden() (on page 8-94)
Data retrieval commands (on page 6-3)

channel.getimage()

This function queries a channel for items associated with that channel when used in a switching operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Related backplane relays Pole settings	Not applicable	channel identifier

Usage

`channels = channel.getimage(channelList)`

channels	A string listing the channels and analog backplane relays associated with the specified items
channelList	A string representing the channels and analog backplane relays to query

Details

The parameter string can contain `"slotX"`(where X equals 1 to 6) or `"allslots"`.

The returned string lists the channels in slot, channel format or slot, row, column format. A request for multiple channels is delimited by a semicolon. Note that commas delimit the specific channels and analog backplane relays for an individual channel in the string.

If an error is detected, the response is `nil`.

An error is generated if:

- A channel pattern is specified
- An empty parameter string is specified
- `slotX` is empty or `allslots` parses to specify no valid channels because all slots are empty

Example 1

```
channel.setpole("2005", 2)
channel.setbackplane("2005", "2911")
channels = channel.getimage("2005")
print(channels)
```

Set channel 5 on slot 2 for a 2-wire switch application.
Associate analog backplane relay 1 in bank 1 on slot 2 with channel 5 on slot 2.
Query channel 5 on slot 2.
Output:
`2005,2911`

Example 2

```
channel.setpole("2003", 4)
channel.setbackplane("2003", "2911,2922")
channels = channel.getimage("2003")
print(channels)
```

Set channel 3 on slot 2 for a 4-wire switch application.
Associate analog backplane relays 1 in bank 1 and 2 in slot 2 with channel 3 on slot 2.
Query channel 3 on slot 2 (assuming channel 3 on slot 2 is on a 40-channel card).
Output:
`2003(2023),2911,2922`

Example 3

`channels = channel.getimage("2003, 2005")` `print(channels)`	Query for channels 2003 and 2005 in a single call (assuming they are configured as shown in examples 1 and 2). Output: `2003(2023),2911,2922;2005,2911`

Example 4

`channels = channel.getimage("2023")` `print(channels)`	Query channel 2023. Query channel 3 on slot 2 (assuming channel 23 on slot 2 is on a 40-channel card). Output: `nil` `2023 is paired for 4-wire operation`

Also see

Channel list notation
channel functions and attributes
channel.pattern.getimage() (on page 8-81)
Data retrieval commands (on page 6-3)

channel.getlabel()

This function retrieves the label associated with one or more channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	slot, row, column or slot, channel identifier

Usage

```
label = channel.getlabel(channelList)
```

label	A string listing the comma-delimited labels for items in *channelList*
channelList	A string listing the channels to query for the label associated with them

Details

The *channelList* parameter can contain more than one channel. If it does, a comma delimits the labels for the channels. The return string lists the labels in the same order that the channels were specified. The *channelList* parameter cannot be an empty string and must be a valid channel.

The *channelList* parameter can contain `slotX` (where X equals 1 to 6) or `allslots`. In this case, the channels are listed before the analog backplane relays.

An error is generated if:

- A specified channel does not exist
- The slot is empty
- The specified channel is not on the installed card
- A channel pattern is specified

Command processing stops as soon as an error is detected, and then a `nil` response is generated. No partial list of labels is returned.

Labels are also supported for digital I/O, DAC, and totalizer channels.

Example

`channel.reset("5001")` `print(channel.getlabel("5001"))` `channel.setlabel("5001", "Device")` `print(channel.getlabel("5001"))`	Reset the channel. Print the default label of the channel. Set the label to "Device". Return the new label. Output: `5001` `Device`

Also see

Channel list notation
channel.setlabel() (on page 8-94)
Data retrieval commands (on page 6-3)

channel.getmatch()

This function gets the match value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0

Usage

```
matchValue = channel.getmatch(channelList)
```

matchValue	Return string listing the comma-delimited match values for channels in *channelList*
channelList	String specifying digital I/O or totalizer channels to query, using normal channel list syntax

Details

If a width greater than 1 is specified with `channel.setmatch()`, the *matchValue* contains the additional channel width specified at set time. For example, the value of 65535 with a width of 2 returns 65535. If the width is 1, 255 is returned.

DAC, backplane, and 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

`print(channel.getmatch("slot6"))`	Query the match values set for digital I/O channels 1 to 5 on slot 6 and totalizer channels 6 to 9 on slot 6, assuming a Model 3750 card. Output (assuming defaults): `0,0,0,0,0,0,0,0,0`

Also see

channel.setmatch() (on page 8-96)

channel.getmatchtype()

Gets the match type.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	4 (channel.MATCH_NONE)

Usage

`matchType = channel.getmatchtype(channelList)`

matchType	Return string listing the comma-delimited states for channels in *channelList*
channelList	String specifying the digital I/O or totalizer channels to query, using normal channel list syntax

Details

The channel match types are:

- 1 for match exactly
- 2 for match any
- 3 for match not exact
- 4 for match none

DAC, backplane, and switch channels are not supported. If these channels are included in a range or slot specifier, they are ignored; otherwise, an error is generated.

Example

`print(channel.getmatchtype("6001:6009"))`	Query the match type for digital I/O channels 1 through 5 and totalizer channels 6 through 9 on slot 6 (assuming a 3750 card). Output: `4,4,4,4,4,4,4,4,4`

Also see

Channel List notation
channel.setmatchtype() (on page 8-97)

channel.getmode()

Gets the present mode attribute for a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0 for digital I/O channels 3 for totalizer channels 49 for DAC channels

Usage

```
mode = channel.getmode(channelList)
```

mode	Return string of a comma-delimited list of modes
channelList	String specifying digital I/O, DAC, or totalizer channels to query, using normal channel list syntax

Details

For digital I/O channels, the following modes are supported:

- channel.MODE_INPUT (default) or 0
- channel.MODE_OUTPUT or 1
- channel.MODE_PROTECT_OUTPUT or 3

For totalizer channels, the following modes are supported:

- channel.MODE_RISING_EDGE or 1
- channel.MODE_FALLING_EDGE or 0
- channel.MODE_RISING_TTL_EDGE (default) or 3
- channel.MODE_FALLING_TTL_EDGE or 2
- channel.MODE_RISING_EDGE_READ_RESET or 5
- channel.MODE_FALLING_EDGE_READ_RESET or 4
- channel.MODE_RISING_TTL_EDGE_READ_RESET or 7
- channel.MODE_FALLING_TTL_EDGE_READ_RESET or 6

For DAC channels, the following modes are supported:

- channel.MODE_VOLTAGE_1 or 17
- channel.MODE_CURRENT_1 or 1
- channel.MODE_CURRENT_2 or 2
- channel.MODE_PROTECT_VOLTAGE_1 (default) or 49
- channel.MODE_PROTECT_CURRENT_1 or 33
- channel.MODE_PROTECT_CURRENT_2 or 34

Switch and analog backplane channels do not have modes. If included in a range or slot specifier, they are ignored. If they are specified directly, an error is generated.

Example

`print(channel.getmode("slot6"))`	Query the configuration of the channels on slot 6. Assuming a 3750, channels 1 to 5 are digital I/O, channels 6 to 9 are totalizers, and channels 10 to 11 are DACs. Output: `0,0,0,0,0,3,3,3,3,49,49`

Also see

Channel list notation
channel.setmode() (on page 8-98)

channel.getoutputenable()

Gets the present output enable attribute for a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0

Usage

```
outputEnable = channel.getoutputenable(channelList)
```

outputEnable	Return string of a comma-delimited list of output enable values
channelList	String specifying DAC channels to query, using normal channel list syntax

Details

For DAC channels, output enable indicates whether or not the DAC is driving the output. Response values are:

- 0: Output enable is OFF
- 1: Output enable is ON

Switch, digital I/O, totalizer, and backplane channels do not have modes. If they included in a range or slot specifier, they are ignored. If they are specified directly, an error is generated.

Example

`print(channel.getoutputenable("slot1"))`	Query the state of all DAC channels on slot 1 (assuming a Model 3750 card, this would be channels 10 and 11). Output: `0,0`

Also see

channel.setoutputenable() (on page 8-100)

channel.getpole()

Queries the pole settings for the specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Card dependent, typically 2

Usage

```
poles = channel.getpole(channelList)
```

poles	Returns a string consisting of the poles, comma separated, based on *channelList*
channelList	A string listing the channels to query for their pole settings

Details

channelList can contain `"slotX"`, where *X* equals 1 to 6, or `"allslots"`.

When the channel list parameter for this function is `"slotX"`, the response first lists the channels starting from lowest to highest.

When the channel list parameter for this function is `"allslots"`, the response starts with slot 1 and increases to slot 6. Each slot is processed completely before going to the next. Keeping this in mind, all slot 1 channels are listed before slot 2 channels.

The response is the numeric value representing the pole selection and not the text. For example, 4-pole selection is 4 and not `channel.POLES_FOUR`.

An error message is generated if:

- An empty parameter string is specified.
- The specified channel does not exist for card installed in slot.
- Parameter syntax error such as incorrect format for `channelList`.
- A channel pattern was specified.
- An analog backplane relay was specified.
- Channel does not support pole setting like a digital I/O.

Command processing stops as soon as an error is detected. No partial list is returned. If an error is detected or the slot is empty, the response is `nil`.

Digital I/O, DAC, backplane, and totalizer channels are not supported.

Example

`channel.reset("slot5")`	Reset the channels on slot 5 only.
`channel.setpole("5003, 5007", 4)`	Set the pole attribute for channels 3 and 7 on slot 5 to be 4.
`polesSlot5 = channel.getpole("5001, 5003, 5005, 5007")`	Query channels 1, 3, 5, and 7 on slot 5 for pole settings.
`print(polesSlot5)`	View the pole attribute for the specified channels. Output: `2,4,2,4`

Also see

Data retrieval commands (on page 6-3)
channel.setpole() (on page 8-101)

channel.getpowerstate()

Gets the current power state attribute for a totalizer or DAC channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	1

Usage

`states = channel.getpowerstate(channelList)`

states	Return string of a comma-delimited list of power states
channelList	String specifying the channels to query, using normal channel list syntax

Details

See card-specific documentation for important potential implications (warm-up times, effective coverage, use cases, and so on) when disabling or enabling power to a channel.

Not all channels can be disabled. If a channel that cannot be disabled is included in a range, it is ignored. If it is specified directly, an error is generated.

Example

`print(channel.getpowerstate("1006"))`	Get the current power state attribute for a totalizer channel 6 of slot 1 (assuming a Model 3750 card). Output (assuming defaults): `1`

Also see

channel.setpowerstate() (on page 8-103)

channel.getstate()

Queries the state indicators of the channels in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Not saved	0

Usage

```
state = channel.getstate(channelList)
state = channel.getstate(channelList, indicatorMask)
```

state	Return string listing the comma-delimited states for the channels in *channelList*
channelList	String specifying the channels to query, using normal channel list syntax
indicatorMask	Value to specify only certain indicators; if omitted, all indicators are returned

Details

Each bit in the *state* represents a different indicator. Therefore, multiple indicators can be present (the OR operation is performed bitwise). All state or state latch commands behave in this manner.

Different channel types support different state information (indicators). The optional state *indicatorMask* can be used to return only certain indicators. If there is no *indicatorMask*, then all indicators are returned.

The following status indicators are defined:

- `channel.IND_CLOSED`
- `channel.IND_OVERLOAD`
- `channel.IND_MATCH`
- `channel.IND_OVERFLOW`

Indicators can be latched or unlatched, depending on other system settings. Latched indicators mean that the condition has occurred since the last reset command (or power cycle). Unlatched indicators mean that the condition occurred when the `channel.getstate()` command was issued. The overflow and overload indicators default to a latched condition.

Although the `channel.getstate()` command returns a string representing a number, this can be easily changed to a number and then compared to one of the provided Lua constants.

For switch channels, the only state information is an indicator of relay state (`channel.IND_CLOSED`).

For digital I/O channels, the state information includes an indicator for the state of auto protection and whether the match value has been matched (`channel.IND_OVERLOAD` and `channel.IND_MATCH`).

For totalizer channels, the state information includes an indicator for overflow and whether the match value has been matched (`channel.IND_OVERFLOW` and `channel.IND_MATCH`).

For DAC channels, the state information includes an indicator for the state of auto protection (`channel.IND_OVERLOAD`).

For more specific information about the overflow and overload indicators, refer to the documentation for the specific card on which the specified channel resides.

Example 1

`channel.close("4005, 4007, 4017, 4003")` `State = channel.getstate("4001:4020")` `print(State)`	Close channels 5, 7, 17, and 3 on slot 4. Query the state of the first 20 channels on slot 4. View the response assigned to State. Output (assuming a Model 3720): `0,0,1,0,1,0,1,0,0,0,0,0,0,0,0,0,1,0,0,0`

Example 2

`PathList = channel.pattern.getimage("Path")` `print(Path)` `print(channel.getstate(Path))`	See the state of channels and analog backplane relays in the channel pattern called `"Path"`. Output: `4003,4005,4007,4017,4911,4922` `1,1,1,1,1,1`

Example 3

`PathState = channel.getstate("Path")` `print(PathState)`	Another way to see the state of channels and analog backplane relays in channel pattern `"Path"` in Example 2 without getting the channels and analog backplane relays first. Output: `1,1,1,1,1,1`

Example 4

`if bit.bitand(channel.IND_OVERLOAD,` `   tonumber(channel.getstate("4010"))) == 1` `then` `   print("OVERLOAD")` `end`	Use the following command to check for an overload on a DAC channel. In the previous example, `channel.getstate()` returns a string that is converted to a number using the Lua `tonumber()` command. `channel.IND_OVERLOAD` equates to the number 2. Because the state is a bit-oriented value, a logical AND operation must be performed on the state and the overload constant to isolate it from other indicators. The `tonumber()` command only works with a single channel. When multiple channels are returned (for example, `channel.getstate("slot4")`), this string must be parsed by the comma delimiter to find each value.

Also see

channel.getclose() (on page 8-61)
channel.setmatch() (on page 8-96)

channel.getstatelatch()

This function gets the mask representing the states that would be latched if they occurred.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	14 for overload, match, and overflow

Usage

```
state = channel.getstatelatch(channelList)
```

state	Return string listing the comma-delimited latch states for channels in *channelList*: • 2: Channel overload • 4: Channel match • 8: Channel overflow
channelList	String specifying the channels to query, using normal channel list syntax

Details

Applicable to digital I/O, totalizer, and DAC channels only.

Each indicator is represented by a bit in the mask.

Example

```
myState = channel.getstatelatch("1001")
print(myState)
```

Queries the state event latch on digital I/O channel 1 in slot 1 assuming a Model 3750.

Example

```
channel.setstatelatch("6010", bit.bitor(channel.IND_OVERFLOW,
   channel.IND_OVERLOAD))
print(channel.getstatelatch("6010"))
```

Generate either an overflow or overload event on DAC channel 10 in slot 6, assuming a Model 3750.

Query for the state latch for channel 10 on slot 6.

Output:

```
10
```

Also see

channel.setstatelatch() (on page 8-104)

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*)

type	Returns a string listing the comma-delimited types for channels in *channelList*
channelList	String specifying the channels to query, using normal *channelList* 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_SWITCH or 1
- channel.TYPE_BACKPLANE or 2
- channel.TYPE_DAC or 8
- channel.TYPE_DIGITAL or 4
- channel.TYPE_TOTALIZER or 16

Refer to the card-specific documentation for more information about the channel types available for your card.

Example 1

`print(channel.gettype("1001, 1911"))`	Query the channel type of channel 1 and analog backplane relay 1 of bank 1 in slot 1, assuming a Model 3720. Output: 1,2

Example 2

`print(channel.gettype("slot6"))`	Query the channel types on slot 6, assuming a Model 3750. Output: 4,4,4,4,4,16,16,16,16,8,8 This shows that channels 1 to 5 are digital I/O types, channels 6 to 9 are totalizer types, and channels 10 and 11 are DAC types.

Also see

None

channel.open()

This function opens the specified channels, analog backplane relays, and channel patterns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.open(channelList)`

channelList	String listing the channels, analog backplane relays, and channel patterns to open

Details

This function opens the specified channels based on the channel's switching configuration.

For the items specified to open, the channels associated with them open, as well as the associated analog backplane relays for each. For channel patterns, the analog backplane relays that get opened are the ones that are specified when the pattern is created (through `channel.pattern.setimage()` and `channel.pattern.snapshot()`). For channels, they are the ones specified with the `channel.setbackplane()` function. Another option for opening analog backplane relays with this command is to include them in the *channelList* parameter.

This command has no effect on how the DMM is configured.

The settling time associated with a channel must elapse before the command completes. User delay is not added when a relay opens.

For digital I/O, DAC, and totalizer channels, there is no valid behavior; calling on a specific channel generates an error. If a digital I/O, DAC, or totalizer channel is in the range of channels, the channel is ignored.

Example 1

`channel.open("1001:1005, 3003, Chans")`	Opens channels 1 to 5 on slot 1, channel 3 on slot 3, and the channel pattern or label `Chans`.

Example 2

`channel.open("slot3, slot5")`	Opens all channels on slots 3 and 5.

Example 3

`channel.open("allslots")`	Opens all channels on all slots.

Also see

channel.close() (on page 8-50)
channel.exclusiveclose() (on page 8-56)
channel.exclusiveslotclose() (on page 8-57)
channel.getclose() (on page 8-61)
channel.getdelay() (on page 8-64)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84)
channel.getstate() (on page 8-75)
channel.setdelay() (on page 8-93)
channel.setforbidden() (on page 8-94)
channel.setbackplane() (on page 8-90)
dmm.close() (on page 8-167)
dmm.open() (on page 8-219)

channel.pattern.catalog()

This function creates a list of the user-created channel patterns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
for name in channel.pattern.catalog() do
   ...
end
```

name	String representing the user-defined name of the channel pattern that is assigned by the catalog function during the `for` loop

Details

This function allows you to print or delete all user-created channel patterns in the run-time environment. The entries that are returned are listed in random order.

Example

```
channel.pattern.setimage("3001,3031",
   "patternA")
channel.pattern.setimage("3002,3032",
   "patternB")
channel.pattern.setimage("3003,3033",
   "patternC")

for name in channel.pattern.catalog() do
   print(name .. " = " ..
      channel.pattern.getimage(name))
   channel.pattern.delete(name)
end
```

This example prints the names and items associated with all user-created channel patterns. It then deletes the channel pattern.

```
patternC = 3003,3033
patternA = 3001,3031
patternB = 3002,3032
```

Also see

channel.pattern.delete() (on page 8-81)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84)

channel.pattern.delete()

This function deletes a channel pattern.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.pattern.delete(`*`name`*`)`

name	A string representing the name of the channel pattern to delete

Details

An error is generated if the name does not exist as a channel pattern.

Example

`channel.pattern.delete("Channels")`	Deletes a channel pattern called `Channels`.

Also see

channel.pattern.catalog() (on page 8-80)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84)

channel.pattern.getimage()

This function queries a channel pattern for associated channels and analog backplane relays.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Pole setting change	Create configuration script Save setup	Not applicable

Usage

`channelList`` = channel.pattern.getimage(`*`name`*`)`

channelList	A string specifying a list of channels and analog backplane relays that are represented by the name
name	A string representing the name of the channel pattern to query

Details

The returned string lists the channels in the slot, column or slot, row, column format, even if a channel pattern was used to create it. Results for multiple channel patterns are delimited by a semicolon (`;`). Commas delimit the specific channels and analog backplane relays in a single channel pattern in the string.

If you change a pole setting for a channel that is associated with a channel pattern, the channel pattern is deleted. Be sure to configure the pole setting for channels (`channel.setpole`) before creating a channel pattern.

Example

```
-- Set up two patterns
channel.pattern.setimage("4001:4005", "myPattern")
channel.pattern.setimage("2001,2003,2005", "myRoute")

-- Print images
myImage = channel.pattern.getimage("myPattern")
print(myImage)
print(channel.pattern.getimage("myRoute"))
print(channel.pattern.getimage("myRoute,
   myPattern"))
```

Using a Model 3721 (or similar model) card in slots 2 and 4, this example creates two channel patterns and then queries these patterns.

Output:

```
4001,4002,4003,4004,4005
2001,2003,2005
2001,2003,2005;4001,4002,4003,4004,4005
```

Also see

channel.pattern.catalog() (on page 8-80)
channel.pattern.delete() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.setpole() (on page 8-101)

channel.pattern.setimage()

This function creates a channel pattern and associates it with the specified name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Pole setting change	Create configuration script Save setup	Not applicable

Usage

`channel.pattern.setimage(`*`channelList`*`, `*`name`*`)`

channelList	A string listing the channels, channel patterns, or analog backplane relays to use when creating the new channel pattern
name	A string representing the name to associate with the new channel pattern

Details

If *name* is used for an existing channel pattern, that pattern is overwritten with the new pattern channel image (if no errors occur). The previous image associated with the name is lost. The DMM configuration associated with the pattern remains unchanged in this scenario.

The channel pattern is not created if an error is detected. You can create a channel pattern with an empty *channelList* parameter. This will create a pattern that has no channels or analog backplane relays associated with it. The behavior of using an empty pattern in a channel list parameter is dependent on the command. For example:

`channel.pattern.setimage("", "Empty_pattern")`	Creates an empty pattern.
`channel.close("Empty_pattern")`	Generates error code 1115, "Parameter error no valid channels in channel list."
`channel.exclusiveslotclose("Empty_pattern")`	Generates error code 1115, "Parameter error no valid channels in channel list."
`channel.open("Empty_pattern")`	Generates error code 1115, "Parameter error no valid channels in channel list."
`channel.exclusiveclose("Empty_pattern")`	Opens any closed channels or analog backplane relays in the instrument.
`channel.close("Empty_pattern, 5005")`	Closes channel 5005.
`channel.exclusiveslotclose("Empty_pattern, 5003")`	Opens any closed channel on slot 5 and closes channel 3 on slot 5.

A channel pattern must include the analog backplane relays and the desired channels. Once a channel pattern is created, the only way to add a channel or analog backplane relay to an existing pattern is to delete the old pattern and recreate the pattern with the new items.

If you change a pole setting for a channel that is associated with a channel pattern, the channel pattern is deleted. Be sure to configure the pole setting for channels (`channel.setpole`) before creating a channel pattern.

Channel patterns are stored when you run the `createconfigscript()` command or `setup.save()` command.

Channel patterns are lost when power is cycled. Use `setup.recall()` or a script created with `createconfigscript()` to restore them.

Including any channels of type digital I/O, DAC, and totalizer generates an error.

The following restrictions exist when naming a channel pattern:

- The name must contain only letters, numbers, or underscores
- The name must start with a letter
- The name is case sensitive

Examples of valid names:

- `Channels`
- `Chans`
- `chans`
- `Path1`
- `Path20`
- `path_3`

Examples of invalid names:

- `1path` (invalid because it starts with a number)
- `my chans` (invalid because it contains a space)
- `My,chans` (invalid because it contains a comma)
- `Path1:10` (invalid because it contains a colon)

An error is generated if:

- The *name* parameter already exists as a label
- Any channel is forbidden to close
- Insufficient memory exists to create the channel pattern
- The parameter string contains `slotX` (where *X* equals 1 to 6) or `allslots`
- The name parameter contains a space character
- The pattern name exceeds 19 characters

Example 1

`channel.pattern.setimage("3001:3010", "Channels")`	For this example, assume there is a Keithley Model 3721 or similar card in slot 3.
`oldList = channel.pattern.getimage("Channels")` `newList = oldList .. ", 3911"` `channel.pattern.delete("Channels")` `channel.pattern.setimage(newList, "Channels")` `channel.open("slot3")` `channel.close("Channels")` `print(channel.getclose("slot3"))`	Create a pattern. Append a channel to the pattern by retrieving the pattern and recreating it. Recreate the pattern with the new image. Open all channels on slot 3 and close the pattern `Channels`. Output: `3001;3002;3003;3004;3005;3006;3007;3008;3009;3010;3911`

Example 2

`channel.pattern.setimage("3001:3010", "Channels")` `channel.open("slot3")` `channel.close("Channels, 3911")` `print(channel.getclose("slot3"))`	An alternate solution to the example above is to create the pattern, then add the analog backplane relay when you close the channel. This eliminates the need to get the image, delete the image and recreate it. Output: `3001;3002;3003;3004;3005;3006;3007;3008;3009;3010;3911`

Also see

createconfigscript() (on page 8-115)
channel.pattern.catalog() (on page 8-80)
channel.pattern.delete() (on page 8-81)
channel.pattern.getimage() (on page 8-81)
channel.pattern.snapshot() (on page 8-84)
channel.setpole() (on page 8-101)
setup.save() (on page 8-370)
setup.recall() (on page 8-370)

channel.pattern.snapshot()

This function creates a channel pattern.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Pole setting change	Create configuration script Save setup	Not applicable

Usage

`channel.pattern.snapshot(name)`

name	A string representing the name to associate with the present state of channels and analog backplane relays

Details

This command stores an image of presently closed channels and analog backplane relays and associates them with the *name* parameter.

If *name* is already used for an existing channel pattern, that pattern is overwritten with the new pattern channel image (if no errors occur). The DMM configuration associated with the pattern remains unchanged.

The following restrictions exist when naming a channel pattern:

- The name must contain only letters, numbers, or underscores
- The name must start with a letter
- The name is case sensitive

Examples of valid names:

- `Channels`
- `Chans`
- `chans`
- `Path1`
- `Path20`
- `path_3`

Examples of invalid names:

- `1path` (invalid because it starts with a number)
- `my chans` (invalid because it contains a space)
- `My,chans` (invalid because it contains a comma)
- `Path1:10` (invalid because it contains a colon)

An error is generated if:

- The *name* parameter already exists as a label
- Insufficient memory exists to save the channel pattern and name in persistent memory
- The pattern name exceeds 19 characters or contains a space

Issuing this function on an existing pattern invalidates the existing scan list (the pattern may or may not be used in the current scan list). Creating a new pattern does not invalidate the existing scan list.

Channels of type DAC, totalizer, and digital I/O are ignored.

Channel patterns are stored when you run the `createconfigscript()` command or `setup.save()` command.

Channel patterns are lost when power is cycled. Use `setup.recall()` or a script created with `createconfigscript()` to restore them.

If you change a pole setting for a channel that is associated with a channel pattern, the channel pattern is deleted. Be sure to configure the pole setting for channels (`channel.setpole`) before creating a channel pattern.

Example

`channel.pattern.snapshot("voltagePath")`	Creates a pattern named `voltagePath` that contains the presently closed channels and analog backplane relays.

Also see

createconfigscript() (on page 8-115)
channel.pattern.catalog() (on page 8-80)
channel.pattern.delete() (on page 8-81)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.setpole() (on page 8-101)
setup.save() (on page 8-370)
setup.recall() (on page 8-370)

channel.read()

This function reads a value from a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = channel.read(channelList)
value = channel.read(channelList, width)
value = channel.read(channelList, width, readingBuffer)
```

value	Return string that lists the comma-delimited read values for the channels in *channelList*
channelList	String that specifies a list of channels, using channel list notation
width	Specifies reading over multiple consecutive channels (default 1)
readingBuffer	Reading buffer to store read values

Details

For digital I/O channels, only a width of 1, 2, 3, or 4 is supported. Any information (bits) greater than the specified *width* is returned as zero. For example, if channels 1 and 2 are both 255, a reading with a *width* of 1 returns 255 and a *width* of 2 with channel 1 returns 65535. Values read from outputs reflect their current setting. If the read channel is in the overload state, the read value is indeterminate.

For widths greater than 1, the specified channel occupies the least significant byte. For example, reading the value 4293844224 (hex ffeedd00) from channel 1 with a *width* of 4 indicates channel 1 is 0 (hex 0), channel 2 is 221 (hex dd), channel 3 is 238 (hex ee), and channel 4 is 255 (hex ff). Reading the value of 0 (hex 0) from channel 1 with a *width* of 1 indicates channel 1 is 0 (hex 0) and other channels are not included.Totalizer and DAC channels do not support a *width* other than 1 and result in an error if specified.

Switch and backplane channels are not supported.

For a channel with a power state of `OFF`, the returned value is `DISABLED`. The value into the reading buffer is indeterminate.

Example

`count = channel.read("1006")`	Read the count from the first totalizer channel (channel 6) in slot 1, assuming a Model 3750.

Also see

None

channel.reset()

This function resets the specified channel list items to factory default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.reset(channelList)`

channelList	A string that lists the items to reset; the string can include: • `allslots` • `slotX`, where *X* is the slot number • channel patterns • channels, including a range of channels • analog backplane relays

Details

For the items specified in *channelList*, the following actions occur:

- Any closed channels and analog backplane relays open
- Any 4-pole channels reset to 2-pole operation and their paired channels are changed to match
- Additional user delay is set to zero (0)
- Labels are removed
- Analog backplane relays specified by the `channel.setbackplane()`function are cleared
- If the channel is forbidden to close, it is cleared from being forbidden to close
- If the channels are used in channel patterns, the channel patterns that contain the channels are deleted.
- The DMM configurations of all channels are set to `nofunction`

Using this function to reset a channel or backplane relay involved in scanning invalidates the existing scan list. The list has to be recreated before scanning again.

For all channels, any trigger settings are removed.

For digital I/O channels, operation is set to input, the match is set to zero (0), and auto-protect is turned on.

For totalizer channels, operation is set to falling edge and TTL level.

For DAC channels, output is turned off and auto-protect is turned on. Operation is set to –12 V to + 12 V.

The rest of the instrument settings are unaffected. To reset the entire system to factory default settings, use the `reset()` command.

Example 1

`channel.reset("allslots")`	Performs a reset operation on all channels on the instrument.

Example 2

`channel.reset("slot1")`	Resets channels on slot 1 only.

Example 3

`channel.reset("3001:3005")`	Resets only channels 1 to 5 on slot 3.

Example 4

`channel.reset("5005, 5915")`	Resets only channel 5 and analog backplane relay 5 in bank 1 on slot 5

Also see

Channel functions and attributes
Channel list notation
channel.setbackplane() (on page 8-90)
dmm.reset() (on page 8-228)
reset() (on page 8-317)
scan.reset() (on page 8-334)

channel.resetstatelatch()

This function resets the channel state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`channel.resetstatelatch(channelList, state)`

channelList	String that specifies the channels that need to have their states reset, using normal channel list syntax
state	String that lists the comma-delimited states for channels in *channelList* that are to have their states reset

Details

This function is applicable to digital I/O, totalizer, and DAC channels only.

The values for *state* are:

- `channel.IND_MATCH` or 4
- `channel.IND_OVERFLOW` or 8
- `channel.IND_OVERLOAD` or 2

Multiple states can be set by performing a logical OR operation on the values.

For *channelList*, use `channel.ALL` to reset all states.

States can be latched or unlatched, depending on other system settings. Latched states indicate that the condition occurred since the last reset (or power cycle). Unlatched states indicate that the condition has occurred when the `channel.getstate()` command was issued. The Overflow and Overload states default to latched.

If states are not cleared using `channel.resetstatelatch()`, you may not be reading the present state of the channel.

If the *state* is reset but the condition that caused the channel state to latch still exists, the state is reset, but a second event is generated through the channel trigger model.

Example

`channel.resetstatelatch("1001", channel.IND_MATCH)`	Clears out a match indicator that was latched on digital I/O channel 1 of slot 1, assuming a Model 3750.

Also see

channel.getstate() (on page 8-75)
channel.getstatelatch() (on page 8-77)
channel.setstatelatch() (on page 8-104)

channel.setbackplane()

This function specifies the list of analog backplane relays to use with the specified channels when they are used in switching applications.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup Pole setting change	Create configuration script Save setup	None

Usage

`channel.setbackplane(`*`channelList`*`, `*`abuslist`*`)`

`channelList`	A string that lists the channels to change
`abuslist`	A string that lists the analog backplane relays to set for the channels specified in *`channelList`*

Details

The parameter string *`channelList`* can contain "slot*X*", where *X* equals 1 to 6, or *`"allslots"`*.

The *`abuslist`* parameter must specify the entire list of analog backplane relays that are needed.

The analog backplane relays specified in the *`abuslist`* parameter are used or affected by:

- `channel.close()`, used during the processing of the command
- `channel.exclusiveclose()`, used during the processing of the command
- `channel.open()`, used during the processing of the command
- `channel.setpole()` clears the analog backplane relays
- `scan.execute()` or `scan.background()`, if channels are configured for switching (the assigned DMM configuration has the function set to `"nofunction"`)

The analog backplane relays specified in the *`abuslist`* parameter are not used or affected by:

- `dmm.close()`
- `dmm.open()`
- `scan.execute()` or `scan.background()` if the channels are configured for measuring (the DMM configuration has the function set to something other than `"nofunction"`).

For channel patterns, the analog backplane relays are specified when the pattern is created (see channel.pattern.getimage() (on page 8-81) and channel.pattern.snapshot() (on page 8-84)). Channel patterns do not have a poles setting associated with them.

If this command is updated, the previous list is replaced with the new specified analog backplane relays in the *`abuslist`* parameter.

For channels, as their pole setting change, the list of analog backplane relays gets cleared. Therefore, after changing the pole settings, send `channel.setbackplane()`with the appropriate analog backplane relay channels. When clearing the backplane channels, this can involve clearing the paired channel, whether pairing or un-pairing channels. For example, on a 40-channel card, channels 1 and 21 are paired when the poles for channel 1 is set to 4. Therefore, setting the poles setting on channel 1 to 4 clears the backplane channels for channels 1 and 21. Likewise, they are both cleared when the poles setting is set back to 2 on channel 1.

Calling this function on an existing channel involved in scanning invalidates the existing scan list.

An error is generated if:

- An empty slot is specified.
- A specified channel or analog backplane relay does not exist for the card installed in a slot.
- An empty parameter string is received for *channelList*. An empty string is allowed for *abuslist*. A parameter string of just spaces is treated like an empty string.
- A specified channel does not have analog backplane relays associated with it, such as digital I/O.
- An analog backplane relay is specified in *channelList*.
- A channel is specified in *abuslist*.
- A channel pattern is specified.

If a syntax error occurs, command processing stops and no changes are made.

Example 1

`channel.setbackplane("2002", "2913, 2914")` `channel.open("allslots")` `channel.close("2002")` `print(channel.getclose("allslots"))`	Use analog backplane relays 3 and 4 in bank 1 of slot 2 for a switching application on channel 2 of slot 2. Open all channels in the instrument. Close channel 2 on slot 2. Query for all closed channels in the instrument. Output (assuming channel 2002 is configured for 2-pole operation): `2002;2913;2914`

Example 2

`print(channel.getbackplane("2002"))`	Query the analog backplane relays for channel 2 of slot 2, assuming the configuration of the previous example. Output: `2913,2914`
`channel.open("slot2")`	Open all channels on slot 2 only.
`channel.setpole("2002", 4)`	Change the poles on channel 2 of slot 2 to 4 (this clears previously assigned backplanes to the channel).
`channel.close("2002")`	Close channel 2 on slot 2.
`print(channel.getclose("slot2"))`	Query for closed channels on slot 2 (note that the backplane relays have been cleared and the paired channel, 2022, is in parentheses) Output: `2002(2022)`
`channel.open("slot2")`	Open all channels on slot 2 only.
`channel.setbackplane("2002", "2911, 2922")`	Assign analog backplane relay 1 of bank 1 and relay 1 of bank 2 of slot 2 to channel 2 of slot 2.
`channel.close("2002")`	Close channel 2 on slot 2.
`print(channel.getclose("slot2"))`	Query for closed channels on slot 2. Output: `2002(2022);2911;2922`

Also see

channel.close() (on page 8-50)
channel.exclusiveclose() (on page 8-56)
channel.getbackplane() (on page 8-59)
channel.open() (on page 8-79)
channel.setpole() (on page 8-101)
scan.background() (on page 8-323)
scan.execute() (on page 8-327)

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 Channel reset Recalls setup	Create configuration script Save setup	0

Usage

`channel.setdelay(channelList, value)`

channelList	A string listing the channels that need modifications to their delay time
value	Desired delay time for items in *channelList*. Minimum is 0 seconds

Details

Setting a delay only applies to switch channels. An error occurs for a read/write channel such as digital input/output. The delay being specified by value may be updated based on a card's resolution for delay. To see if the delay value was modified after setting, use the `channel.getdelay()` command to query.

Channel patterns get their delay from the channels that comprise the pattern. Therefore, specify the delay for a pattern through the channels. A pattern incurs the longest delay of all channels comprising that pattern.

An error message is generated if:

- The value is an invalid setting for the specified channel
- A channel pattern is specified
- The channel is for an empty slot
- An analog backplane relay is specified.

Command processing will stop as soon as an error is detected and no delay times will be modified.

NOTE

Pseudocards do not replicate the additional delay time.

Example 1

`channel.setdelay("5001, 5003" , 50e-6)`	Sets channels 1 and 3 on slot 5 for a delay time of 50 microseconds.

Example 2

`channel.setdelay ("slot3", 0)`	Sets the channels on slot 3 for 0 delay time.

Also see

channel.getdelay() (on page 8-64)

channel.setforbidden()

This function prevents the closing of specified channels and analog backplane relays.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Not forbidden

Usage

`channel.setforbidden(channelList)`

channelList	A string that lists the channels and analog backplane relays to make forbidden to close

Details

The *channelList* parameter indicates the scope of channels affected and may include:

- `allslots` or `slotX` (where *X* equals 1 to 6)
- Channel ranges or individual channels
- Analog backplane relays

This function prevents all items contained in the channel list parameter from closing. It applies the "forbidden to close" attribute to the specified channels. To remove the "forbidden to close" attribute, use `channel.clearforbidden()`.

If a channel that is being set to forbidden is used in a channel pattern, the channel pattern is deleted when the channel or analog backplane relay is set to forbidden. Note that if the *channelList* parameter includes a channel pattern, the channel pattern will be deleted when the channels in the patterns are successfully set to forbidden to close.

The channels or analog backplane relays in the *channelList* parameter must be installed in the instrument.

If the scan list contains a channel or analog backplane relay that is forbidden, the scan list is invalidated.

Example

`channel.setforbidden("2002,2004,2006,2008")`	Marks channels 2, 4, 6, and 8 of slot 2 as forbidden to close.
`channel.setforbidden("slot3")`	Marks all channels and analog backplane relays on slot 3 as forbidden to close.

Also see

channel.clearforbidden() (on page 8-49)
channel.getforbidden() (on page 8-66)

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 Channel reset Recall setup	Create configuration script Save setup	No label

Usage

`channel.setlabel(channelList, label)`

channelList	A string that lists the channel to which to set the label
label	A string that represents the label for the channel in *channelList*, up to 19 characters

Details

This command sets the label of the channel specified in *channelList* to the value specified in the *label* parameter. The channel attributes associated with each channel remain unchanged except for their labels.

The *label* parameter must be unique. In addition, it cannot be the same as the name of a channel pattern. If you specify a label that already exists, an error message is generated that indicates a parameter error and channel that that is already associated the specified label.

For example, channel one on slot 4 has a label of `start`. If you send `channel.setlabel("5001", "start")`, error code 1115, "Parameter error label already used with channel 4001," is generated.

To clear the label, set *label* to an empty string (`""`) or to a string with a space as the first character.

After defining a label, you can use it to specify the channel instead of using the channel specifier.

An error is generated if:

- The card in the channel slot does not support a label setting
- The label contains a space; however, if the first character is a space, the label is cleared
- The label is already used to represent a channel pattern

The label does not persist through a power cycle.

Example 1

`channel.setlabel("3001", "start")` `channel.close("start")` `print(channel.getclose("allslots"))`	Sets the label for channel 1 on slot 3 to "start" and closes "start". Output: `3001`

Example 2

`channel.setlabel("3001", "")`	Clears the label for channel 1 on slot 3 back to "3001".

Example 3

`channel.setlabel("3001", " ")`	Also clears the label for channel 1 slot 3 back to "3001".

Also see

Channel list notation
channel functions and attributes

channel.setmatch()

This function sets the match value on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0

Usage

channel.setmatch(*channelList*, *matchValue*)

channel.setmatch(*channelList*, *matchValue*, *mask*)

channel.setmatch(*channelList*, *matchValue*, *mask*, *width*)

channelList	String that specifies the channels to query, using normal channel list syntax
matchValue	Channel value to compare on the specified channel
mask	Value to specify the bits used to mask *matchValue*
width	Value that specifies matches over multiple consecutive channels (default 1)

Details

A bitwise AND operation is performed on *mask* and *matchValue* to determine the final match value used on the channel.

The default mask is `channel.ALL` (all bits).

For digital I/O channels, a *width* of 1, 2, 3, or 4 channels is supported. Any bits greater than the specified *width* are ignored. If a *width* crosses channels, the match status indicator is only on the channel specified in the match value. For example, setting a value with a 2 *width* on channel 3 drives the indicator on channel 3, not channel 4. Match values for output channels are ignored.

Totalizer and DAC channels only support a *width* of 1, and *mask* is ignored.

Switch and backplane 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 1

```
channel.setmatchtype("1001",
   channel.MATCH_EXACT)
channel.setmatch("1001", 32)
```

Generates a match state event on bit B6 of digital I/O channel 1, assuming a Model 3750.

Example 2

```
channel.setmatchtype("6007",channel.MATCH_EXACT)
channel.setmatch("6007", 300)
```

Assuming a model 3750, configure the totalizer channel 7 on slot 6 to generate a match state event when it reaches 300.

Also see

channel.getmatch() (on page 8-69)

channel.setmatchtype()

This function sets the match type on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	4 (channel.MATCH_NONE)

Usage

`channel.setmatchtype(channelList, type)`

channelList	String specifying the channels to set, using normal channel list syntax
type	A value for setting the match operation used on this specific channel

Details

There are four types of match values:

- `channel.MATCH_EXACT` or 1
- `channel.MATCH_ANY` or 2
- `channel.MATCH_NOT_EXACT` or 3
- `channel.MATCH_NONE` or 4

For an EXACT match, the state match indicator only becomes TRUE when the match value AND match mask value EQUAL the channel read value.

For an ANY match, the state match indicator only becomes TRUE when the match value OR match mask value EQUAL the channel read value.

For a NOT_EXACT match, the state match indicator only becomes TRUE when the match value AND match mask value AND channel read value are NOT EQUAL to the match value AND match mask value AND last channel read value. In other words, the match value should be the current value, and if the value changes at all away from the original value, then a match is declared.

For NONE, matching is effectively disabled. This is the default.

For totalizer channels, only `MATCH_EXACT` and `MATCH_NONE` are supported.

This command is not supported on DAC, backplane, and switch channels.

Example

`channel.setmatchtype("1001", channel.MATCH_EXACT)`	Assuming a Model 3750, defines the match type for digital I/O channel 1 to be a `MATCH_EXACT` type.

Also see

channel.getmatchtype() (on page 8-70)

channel.setmode()

This function sets the mode attribute on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Digital I/O: 0 (channel.MODE_INPUT) Totalizer: 3 (channel.MODE_RISING_TTL_EDGE) DAC: 49 (channel.MODE_PROTECT_VOLTAGE_1)

Usage

```
channel.setmode(channelList, mode)
```

channelList	String specifying the channels to set, using normal channel list syntax
mode	The value that sets the mode of a channel's operation

Details

Different channel types contain additional configurable settings. These settings are grouped together by channel type as described in the following paragraphs.

For digital I/O channels, the mode indicates the direction of the channel (input or output). The following modes are supported:

- `channel.MODE_INPUT` (default) or `0`
- `channel.MODE_OUTPUT` or 1
- `channel.MODE_PROTECT_OUTPUT` or 3

For totalizer channels, the mode indicates the configuration of the channel (edge and reset). The following modes are supported:

- `channel.MODE_RISING_EDGE` or 1
- `channel.MODE_FALLING_EDGE` or `0`
- `channel.MODE_RISING_TTL_EDGE` (default) or 3
- `channel.MODE_FALLING_TTL_EDGE` or 2
- `channel.MODE_RISING_EDGE_READ_RESET` or 5
- `channel.MODE_FALLING_EDGE_READ_RESET` or 4
- `channel.MODE_RISING_TTL_EDGE_READ_RESET` or 7
- `channel.MODE_FALLING_TTL_EDGE_READ_RESET` or 6

For DAC channels, the mode indicates the output of the channel (function and range). The output is switched off before any mode change is made, and remains off after the mode has changed. The following modes are supported:

- `channel.MODE_VOLTAGE_1` or 17
- `channel.MODE_CURRENT_1` or 1
- `channel.MODE_CURRENT_2` or 2
- `channel.MODE_PROTECT_VOLTAGE_1` (default) or 49
- `channel.MODE_PROTECT_CURRENT_1` or 33
- `channel.MODE_PROTECT_CURRENT_2` or 34

Changing the mode setting can impact the power consumption of the card. The instrument verifies that power is available before changing the mode. If an insufficient power capability exists, the command generates an error.

Consult the card-specific documentation for more detailed information on mode settings and functionality.

For digital I/O channels, changing the mode from input to output or from output to input adds an additional channel delay (`channel.setdelay()`).

For switch and backplane channels, there is no valid mode setting. Setting a mode on a specific switch or backplane channel generates an error. If the switch or backplane channel is in the range of channels, the switch or backplane channel is ignored.

The specified channel list must use only one channel type. For example, channel list "1001:1004" is only valid if channels 1, 2, 3, and 4 are of the same type. If channel 3 is a different type of channel, the channel list is invalid and an error is generated.

Example

`channel.setmode("6003:6005", channel.MODE_OUTPUT)`	Assuming a Model 3750, set digital I/O channels 3 to 5 to be configured for output on slot 6, assuming a Model 3750.
`channel.setmode("6007", channel.MODE_FALLING_TTL_EDGE)`	Set the totalizer channel 7 on slot 6 to count the falling edges of a TTL signal.

Also see

channel.setdelay() (on page 8-93)

channel.setoutputenable()

This function sets the output enable attribute on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	0 (channel.OFF)

Usage

```
channel.setoutputenable(channelList, state)
```

channelList	String specifying the channels to set, using normal channel list syntax
state	A value representing the desired state of the channel's output

Details

For DAC channels, output enable indicates whether or not the DAC is driving the output. The following possible states are supported:

- `channel.ON`
- `channel.OFF` (default)

For DAC channels, changing the output state to ON adds an additional channel delay to `channel.setdelay()`.

Channels with output set to OFF consume less power.

Changing the output setting impacts the power consumption of the card. The instrument verifies that power is available before changing the mode. If there is insufficient power capability, the command generates an error. Consult the specific card documentation for information on a channel's output characteristics.

For switch, backplane, digital I/O, and totalizer channels, there is no valid output enable value. Setting output enable on a specific channel generates an error. If one of these channels is in the range of channels, the channel is ignored.

Example

`channel.setoutputenable("1010", channel.OFF)`	Assuming a Model 3750, turns the output off on the first DAC channel (channel 10) in slot 1.

Also see

channel.getoutputenable() (on page 8-72)
channel.setdelay() (on page 8-93)

channel.setpole()

This function specifies the pole setting for a list of channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Card dependent, but typically 2 (channel.POLES_TWO)

Usage

```
channel.setpole(channelList, value)
```

channelList	String that specifies a list of channels, using channel list notation
value	Desired pole setting for the channels in *channelList*. Use the following: • For one-pole: `channel.POLES_ONE` or `1` • For two-pole: `channel.POLES_TWO` or `2` • For four-pole: `channel.POLES_FOUR` or `4`

Details

The parameter string can contain `allslots` or `slotX`, where *X* equals 1 to 6.

Channel patterns do not have a pole setting associated with them. For channel patterns, the pole setting indicates if the paired channel should be used when the pattern is created and the analog backplane relays must be specified when creating the pattern (with `channel.pattern.setimage()` and `channel.pattern.snapshot()`). Channel patterns get deleted as the pole settings of the channels in the pattern image get changed.

You manipulate the analog backplane relays after setting the desired pole setting by using the `channel.setbackplane()` function for channels. For channels, as the pole setting changes, their analog backplane relays, specified by `channel.setbackplane()`, get cleared. Therefore, after a pole setting change, you need to add the desired analog backplane relays for the desired pole setting by using `channel.setbackplane()`.

When clearing the backplane channels, this can involve clearing the paired channel, whether pairing or unpairing channels. For example, on a 40-channel card, channels 1 and 21 are paired when the pole setting for channel 1 is set to 4. Therefore, when changing the pole setting on channel 1 to 4, the backplane channels for channels 1 and 21 are cleared. Likewise, they both are cleared when the pole setting is set back to 2 on channel 1.

Calling this function on an existing channel involved in scanning invalidates the existing scan list.

An error message is generated for the following reasons:

- An empty parameter string is specified.
- The value parameter is an invalid setting for the specified channel.
- The specified channel does not exist for the card installed in a slot.
- The channel is for an empty slot.
- The value parameter is invalid for command – parameter out of range error.
- A channel pattern or analog backplane relay was specified.

Command processing stops as soon as an error is detected and no pole settings are modified.

Example 1

`channel.setpole("5001, 5003", channel.POLES_FOUR)`	Sets channels 1 and 3 on slot 5 to four-pole.

Example 2

`channel.reset("slot2")`	Assuming a Model 3721, reset channels on slot 2 only.
`channel.setpole("2001, 2003", channel.POLES_FOUR)`	Set channels 1 and 3 on slot 2 to 4-pole.
`channel.close("2001, 2003")`	Close channels 1 and 3 on slot 2.
`print(channel.getclose("slot2"))`	Query slot 2 for closed channels and analog backplane relays. Output: `2001(2021);2003(2023)` Note that the channels in parentheses are the paired channels because they are in a 4-pole configuration.
`channel.open("slot2")`	Open all channels and analog backplane relays on slot 2.
`channel.setbackplane("2001", "2915")`	Associate analog backplane relay 5 in bank 1 of slot 2 with channel 1 on slot 2.
`channel.setbackplane("2003", "2925")`	Associate analog backplane relay 5 in bank 2 of slot 2 with channel 3 on slot 2.
`channel.close("2001, 2003")`	Close channels 1 and 3 on slot 2.
`print(channel.getclose("slot2"))`	Query slot 2 for closed channels and analog backplane relays. Output: `2001(2021);2003(2023);2915;2925`
`print(channel.getimage("2001, 2003"))`	Query for channels and analog backplane relays that are manipulated when open and close channels 1 and 3 on slot 2. Output: `2001(2021),2915;2003(2023),2925`
`channel.open("slot2")` `channel.setpole("slot2", 2)` `print(channel.getimage("2001, 2003"))`	Clear paired channels and analog backplane relays. Output: `2001;2003` Note that channels are no longer paired or have analog backplane relays associated with them.

Also see

channel.getbackplane() (on page 8-59)
channel.getpole() (on page 8-73)
channel.pattern.setimage() (on page 8-82)
channel.pattern.snapshot() (on page 8-84))
channel.setbackplane() (on page 8-90)

channel.setpowerstate()

This function sets the power state on a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Dependent on installed card, but usually 1 (channel.ON)

Usage

```
channel.setpowerstate(channelList, state)
```

channelList	String that specifies a list of channels, using channel list notation
state	• `channel.OFF` or `0`: Disable the power • `channel.ON` or `1`: Enable the power

Details

When a channel that was previously off is turned on, the channel attributes are reset to their default values (except the power state attribute).

Changing the output setting impacts the power consumption of the card. Channels with an off power state consume less power. Before enabling power, the instrument verifies that power is available before changing the state. If insufficient power capability exists, the command generates an error.

Consult the specific card documentation for information on a channel's power usage characteristics, including default state, possible warmup issues, especially for DAC channels, and effects on other channels.

When a channel with an off power state is used in a scan, results are undefined. No error notification is issued.

For switch, backplane, and digital I/O channels, there is no valid power state attribute. Setting the power state on a specific channel generates an error.

On some cards for totalizer channels, setting the power state of a single channel can affect the power state of other channels. If a single totalizer channel is turned on, all totalizer channels are reset to their defaults.

Example

`channel.setpowerstate("1010", channel.ON)`	Sets the power state for DAC channel 10 on the card in slot 1 to ON, assuming a Model 3750.

Also see

channel.getpowerstate() (on page 8-74)

channel.setstatelatch()

This function sets the state indicators to either latching or nonlatching.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	14 for overload, match, and overflow

Usage

`channel.setstatelatch(channelList, stateLatchMask)`

channelList	String that specifies a list of channels, using channel list notation
stateLatchMask	A value specifying the indicators to latch: • `channel.IND_MATCH` or 4 • `channel.IND_OVERFLOW` or 8 • `channel.IND_OVERLOAD` or 2

Details

Applicable to digital I/O, totalizer, and DAC channels only.

Each indicator is represented by a bit in the mask.

For nonlatching applications, the state indicator clears automatically when the causing condition clears itself. For latching applications, the condition is cleared using the `channel.resetstatelatch()` command.

When using the trigger model, events are always nonlatching (or pulse oriented). However, in latching operation, the event is only generated once at the beginning. In nonlatching operation, the event is generated anytime the condition begins.

Set multiple states by performing a logical `OR` operation on the values.

Example 1

`channel.setstatelatch("1001", channel.IND_MATCH)`	Generate only a match state event on digital I/O channel 1 in slot 1, assuming a Model 3750.

Example 2

```
channel.setstatelatch("6010", bit.bitor(channel.IND_OVERFLOW,
    channel.IND_OVERLOAD))
print(channel.getstatelatch("6010"))
```

Generate either an overflow or overload event on DAC channel 10 in slot 6, assuming a Model 3750.
Query for the state latch for channel 10 on slot 6.
Output:
`10`

Also see

channel.getstate() (on page 8-75)
channel.getstatelatch() (on page 8-77)
channel.resetstatelatch() (on page 8-89)

channel.trigger[N].clear()

This function clears any pending events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.trigger[N].clear()
```

N	Number indicating the trigger line to clear (1 to 8)

Details

This function clears any pending events for the channel trigger specified by *N*.

Example

`channel.trigger[1].clear()`	Clears any pending events on channel trigger 1

Also see

channel.trigger[N].set() (on page 8-107)

channel.trigger[N].EVENT_ID

This constant indicates the trigger event generated by the channel trigger *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
X = channel.trigger[N].EVENT_ID
```

X	The trigger event number
N	Number indicating the channel trigger event ID (1 to 8)

Example

```
scan.trigger.channel.stimulus = channel.trigger[1].EVENT_ID
```

Use channel trigger 1 events to pace the channel action of the scanning or set the trigger stimulus of the channel event detector to channel trigger 1.

Also see

channel.trigger[N].set() (on page 8-107)

channel.trigger[N].get()

This function gets the channel status trigger information that is used to watch the state of a specific channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Empty channel list State match 0

Usage

channelList, *stateMatch* = channel.trigger[*N*].get()

channelList	Returns a string specifying the channels watched by this trigger
stateMatch	Returns a value specifying the state to match when triggering an event
N	Number indicating the channel trigger to get (1 to 8)

Details

This command works for DAC, digital I/O and totalizer channels. Switch channels are not supported.

Example

```
channel.trigger[1].set("1010", channel.IND_MATCH)
chan_list, state_match = channel.trigger[1].get()
print(chan_list, state_match)
```

Assuming a Model 3750, defines channel trigger event 1 to occur when totalizer channel 10 matches its defined match value.

Query for the channels and state conditions associated with channel trigger 1.

Output:

```
1010    4.000000000e+00
```

Example

```
channel.trigger[5].set("6003, 6005",
    channel.IND_MATCH)
print(channel.trigger[5].get())
```

Assuming a Model 3750 card, define a channel trigger event 5 to occur when either digital I/O channel 3 or 5 on slot 6 match their defined values.

View the trigger information associated with channel trigger 5.

Output:

```
6003,6005
    4.000000000e+00
```

Also see

channel.trigger[N].set() (on page 8-107)

channel.trigger[N].set()

This function sets the channel status trigger model to watch the state of a specific channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	Empty channel list State match 0

Usage

`channel.trigger[N].set(channelList, stateMatch)`

channelList	String that specifies a list of channels, using channel list notation
stateMatch	Value specifying the status to match when triggering an event
N	Number indicating the channel trigger to set (1 to 8)

Details

If the channel list contains more than one channel, the trigger acts as a logical OR. When any one of the channels in the list matches the desired state, a trigger event is generated. Therefore, if an indicator is present in both the match and the actual state, an event is triggered. If the match contains more than one state indicator, only one of those indicators needs to be present to trigger the event.

There are a total of eight channel trigger events per Series 3700A, defined by *N*. Using this mechanism, a trigger can be generated when a pattern is matched on an I/O, a totalizer matches a defined count, or an I/O has an overcurrent condition.

Latching functionality is not supported.

This command works for DAC, digital I/O and totalizer channels. Switch channels are not supported.

To clear a trigger that is no longer needed, pass an empty channel list (`""` or `" "`) .

Example 1

`channel.trigger[1].set("1001", channel.IND_MATCH)`	Assuming a Model 3750, defines a channel trigger event 1 to occur when digital I/O channel 1 matches its defined match value.

Example 2

```
channel.trigger[5].set("6003, 6005",
   channel.IND_MATCH)
print(channel.trigger[5].get())

channel.trigger[5].set(" ", channel.IND_MATCH)
print(channel.trigger[5].get())
```

Assuming a Model 3750 card, define a channel trigger event 5 to occur when either digital I/O channel 3 or 5 on slot 6 match their defined values.

View the trigger information associated with channel trigger 5.

Clear the trigger information associated with channel trigger 5.

View the trigger information associated with channel trigger 5.

Output:

```
6003,6005
    4.000000000e+000
 0.000000000e+000
```

Also see

channel.trigger[N].get() (on page 8-106)

channel.trigger[N].wait()

This function waits for the desired trigger or timeout period, whichever comes first.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = channel.trigger[N].wait(timeout)
```

triggered	Returns an indication that a trigger occurred
N	Number indicating the channel trigger to wait for (1 to 8)
timeout	The number of seconds to wait

Details

If one or more trigger events were detected since the last time `channel.trigger[N].wait` or `channel.trigger[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.

The value for timeout must be greater than zero and less than 10,000.

Example

`channel.trigger[1].wait(5)`	Wait 5 seconds for channel trigger event 1 to occur or timeout if trigger event is not detected in 5 seconds.

Also see

None

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)
channel.write(channelList, value, width)
```

channelList	String that specifies a list of channels, using channel list notation
value	The value to be written to the channel (must be decimal value)
width	Value that specifies the channel width of the write

Details

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.

NOTE

You must use decimal values when sending commands to the Series 3700A.

For digital I/O channels, only widths of 1, 2, 3, or 4 are supported. All other widths are ignored. Values written to inputs are ignored. If no specified channel is set for output, then an error is generated. If a width crosses channels, then only the channels set to output are affected.

Totalizers, DACs, and switch channels do not support a width other than 1. Specifying a width greater than 1 results in an error.

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 desired floating point voltage or current. Also, an error is generated if the value is out of range. No action is taken on any channel in the specified channel list.

For digital I/O channels, the value becomes the settings of the digital output.

For totalizer channels, the value becomes the new current totalizer count.

The time it takes to execute the write command is affected by the channel delay setting.

Example

`channel.write("1001", 33)` `channel.write("1006", 0)`	Output a value of 33 to digital I/O channel 1. Set totalizer channel 6 on slot 1 (assuming a Model 3750 card) to 0.

Also see

None

comm.gpib.enable

This attribute describes whether or not communication using the GPIB connection is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.gpib.enable
comm.gpib.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This performs the same function as the **MENU** > **GPIB** > **ENABLE** option that is available through the front panel of the instrument.

Also see

Set the GPIB address (on page 2-59)

comm.lan.enable

This attribute controls whether or not any communication using the LAN connection is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.lan.enable
comm.lan.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This is the master control setting. When this is true (enabled), you may individually control web, Telnet , VXI-11 and raw socket access to the instrument. However, when this is false (disabled), all LAN communication is disabled and this overrides the individual LAN enabled settings.

To disable only certain LAN communication with the instrument, enable this attribute and set the specific LAN communication attribute to false for raw sockets, Telnet, VXI-11 or web.

Example

`comm.lan.enable = false`	Disable all LAN communication with instrument.

Also see

comm.lan.rawsockets.enable (on page 8-111)
comm.lan.telnet.enable (on page 8-112)
comm.lan.vxi11.enable (on page 8-113)
comm.lan.web.enable (on page 8-114)

comm.lan.rawsockets.enable

This attribute describes whether or not communication using raw socket is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.lan.rawsockets.enable
comm.lan.rawsockets.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This performs the same function as the **MENU** > **LAN** > **ENABLE** > **RAW** option available through the front panel of the instrument.

Example

`comm.lan.enable = true` `comm.lan.rawsockets.enable = false`	Enable all LAN communication with instrument, then disable only raw sockets over the LAN.

Also see

comm.lan.enable (on page 8-110)
comm.lan.telnet.enable (on page 8-112)
comm.lan.vxi11.enable (on page 8-113)
comm.lan.web.enable (on page 8-114)
lan.status.port.rawsocket (on page 8-279)
Raw socket connection

comm.lan.telnet.enable

This attribute describes whether or not communication using Telnet is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.lan.telnet.enable
comm.lan.telnet.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This performs the same function as the **MENU** > **LAN > ENABLE > TELNET** option that is available through the front panel of the instrument.

Example

`comm.lan.enable = true` `comm.lan.telnet.enable = false`	Enable all LAN communication with instrument, then disable only Telnet over the LAN.

Also see

comm.lan.enable (on page 8-110)
comm.lan.rawsockets.enable (on page 8-111)
comm.lan.vxi11.enable (on page 8-113)
comm.lan.web.enable (on page 8-114)
lan.status.port.telnet (on page 8-279)
Telnet connection

comm.lan.vxi11.enable

This attribute describes whether or not communication using a VXI-11 connection is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.lan.vxi11.enable
comm.lan.vxi11.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This performs the same function as the **MENU > LAN** > **ENABLE** > **VXI11** option that is available through the front panel of the instrument.

Example

`comm.lan.enable = true` `comm.lan.vxi11.enable = false`	Enable all LAN communication with instrument, then disable only VXI-11 over the LAN.

Also see

comm.lan.enable (on page 8-110)
comm.lan.rawsockets.enable (on page 8-111)
comm.lan.telnet.enable (on page 8-112)
comm.lan.web.enable (on page 8-114)
lan.status.port.vxi11 (on page 8-280)
VXI-11 connection

comm.lan.web.enable

This attribute describes whether or not communication using the web interface is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	true (enabled)

Usage

```
state = comm.lan.web.enable
comm.lan.web.enable = state
```

state	`true`: Enabled `false`: Disabled

Details

This performs the same function as the **MENU** > **LAN** > **ENABLE** > **WEB** option that is available through the front panel of the instrument.

Example

`comm.lan.enable = true` `comm.lan.web.enable = false`	Enable all LAN communication with instrument, then disable only web communication over the LAN.

Also see

comm.lan.enable (on page 8-110)
comm.lan.rawsockets.enable (on page 8-111)
comm.lan.telnet.enable (on page 8-112)
comm.lan.vxi11.enable (on page 8-113)
Web connection

createconfigscript()

This function captures the present settings of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
createconfigscript(scriptName)
```

scriptName	A string that represents the name of the script that will be created

Details

If *scriptName* is set to `autoexec`, the autoexec script in the instrument is replaced by the new configuration script.

If *scriptName* is set to the name of an existing script, the existing script is overwritten.

Once created, the configuration script can be run and edited like any other script.

Example

`createconfigscript("August2013")`	Captures the present settings of the instrument into a script named `August2013`.

Also see

Create a configuration script (on page 2-102)
Save the present configuration (on page 2-100)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of `true` or `false` based on the success of the function
value	The data item to add; *value* can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node (you can only use the *timeout* value while adding data to the local data queue).

The *timeout* value is ignored if the data queue is not full.

The `dataqueue.add()` function returns `false`:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

`dataqueue.clear()` `dataqueue.add(10)` `dataqueue.add(11, 2)` `result = dataqueue.add(12, 3)` `if result == false then` `   print("Failed to add 12 to the dataqueue")` `end` `print("The dataqueue contains:")` `while dataqueue.count > 0 do` `   print(dataqueue.next())` `end`	Clear the data queue. Each line adds one item to the data queue. Output: `The dataqueue contains:` `1.00000e+01` `1.10000e+01` `1.20000e+01`

Also see

dataqueue.CAPACITY (on page 8-116)
dataqueue.clear() (on page 8-117)
dataqueue.count (on page 8-117)
dataqueue.next() (on page 8-118)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable that is assigned the value of `dataqueue.CAPACITY`

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

`MaxCount = dataqueue.CAPACITY` `while dataqueue.count < MaxCount do` `   dataqueue.add(1)` `end` `print("There are " .. dataqueue.count` `   .. " items in the data queue")`	This example fills the data queue until it is full and prints the number of items in the queue. Output: `There are 128 items in the data` `   queue`

Also see

dataqueue.add() (on page 8-115)
dataqueue.clear() (on page 8-117)
dataqueue.count (on page 8-117)
dataqueue.next() (on page 8-118)

dataqueue.clear()

This function clears the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dataqueue.clear()
```

Details

This function forces all `dataqueue.add()` commands that are in progress to time out and deletes all data from the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

```
There are 128 items in the data
    queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 8-115)
dataqueue.CAPACITY (on page 8-116)
dataqueue.count (on page 8-117)
dataqueue.next() (on page 8-118)

dataqueue.count

This attribute contains the number of items in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle	Not saved	Not applicable

Usage

```
count = dataqueue.count
```

count	The number of items in the data queue

Details

The count gets updated as entries are added with `dataqueue.add()` and read from the data queue with `dataqueue.next()`. It is also updated when the data queue is cleared with `dataqueue.clear()`.

A maximum of `dataqueue.CAPACITY` items can be stored at any one time in the data queue.

Example

`MaxCount = dataqueue.CAPACITY` `while dataqueue.count < MaxCount do` `   dataqueue.add(1)` `end` `print("There are " .. dataqueue.count` `   .. " items in the data queue")` `dataqueue.clear()` `print("There are " .. dataqueue.count` `   .. " items in the data queue")`	This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again. Output: `There are 128 items in the data queue` `There are 0 items in the data queue`

Also see

dataqueue.add() (on page 8-115)
dataqueue.CAPACITY (on page 8-116)
dataqueue.clear() (on page 8-117)
dataqueue.next() (on page 8-118)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is `nil`.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

`dataqueue.clear()` `for i = 1, 10 do` `   dataqueue.add(i)` `end` `print("There are " .. dataqueue.count` `   .. " items in the data queue")` `while dataqueue.count > 0 do` `   x = dataqueue.next()` `   print(x)` `end` `print("There are " .. dataqueue.count` `   .. " items in the data queue")`	Clears the data queue, adds ten entries, then reads the entries from the data queue. Note that your output may differ depending on the setting of `format.asciiprecision`. Output: `There are 10 items in the data queue` `1.0000000e+00` `2.0000000e+00` `3.0000000e+00` `4.0000000e+00` `5.0000000e+00` `6.0000000e+00` `7.0000000e+00` `8.0000000e+00` `9.0000000e+00` `1.0000000e+01` `There are 0 items in the data queue`

Also see

dataqueue.add() (on page 8-115)
dataqueue.CAPACITY (on page 8-116)
dataqueue.clear() (on page 8-117)
dataqueue.count (on page 8-117)
format.asciiprecision (on page 8-255)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`delay(seconds)`

seconds	The number of seconds to delay (1 to 100,000 s)

Details

You cannot set a delay for zero seconds.

The instrument delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the instrument to delay 5 μs to 10 μs (typical) more than the requested delay.

Example 1

`beeper.beep(0.5, 2400)` `delay(0.250)` `beeper.beep(0.5, 2400)`	Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.

Example 2

`dataqueue.clear()` `dataqueue.add(35)` `timer.reset()` `delay(0.5)` `dt = timer.measure.t()` `print("Delay time was " .. dt)` `print(dataqueue.next())`	Clear the data queue, add 35 to it, and then delay 0.5 seconds before reading it. Output: `Delay time was 0.500099` `3.500000000e+01`

Also see

None

digio.readbit()

This function reads one digital I/O line. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

data = digio.readbit(*N*)

data	The state of the I/O line
N	Digital I/O line number to be read (1 to 14)

Details

A returned value of zero (0) indicates that the line is low. A returned value of one (1) indicates that the line is high.

Example

`print(digio.readbit(4))`	Assume line 4 is set high, and it is then read. Output: `1.00000e+00`

Also see

digio.readport() (on page 8-120)
digio.writebit() (on page 8-129)
digio.writeport() (on page 8-130)
Digital I/O port (on page 2-28, on page 3-43)

digio.readport()

This function reads the digital I/O port. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readport()
```

data	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the I/O port. The least significant bit (bit B1) of the binary number corresponds to line 1; bit B14 corresponds to line 14.

For example, a returned value of 170 has a binary equivalent of 00000010101010, which indicates that lines 2, 4, 6, and 8 are high (1), and the other 10 lines are low (0).

Example

`data = digio.readport()` `print(data)`	Assume lines 2, 4, 6, and 8 are set high when the I/O port is read. Output: `1.70000e+02` This is binary 10101010

Also see

digio.readbit() (on page 8-120)
digio.writebit() (on page 8-129)
digio.writeport() (on page 8-130)
Digital I/O port (on page 2-28, on page 3-43)

digio.trigger[N].assert()

This function asserts a trigger on one of the digital I/O lines. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

N	Digital I/O trigger line (1 to 14)

Details

The set pulsewidth determines how long the trigger is asserted.

Example

`digio.trigger[2].assert()`	Asserts a trigger on digital I/O line 2.

Also see

digio.trigger[N].pulsewidth (on page 8-125)

digio.trigger[N].clear()

This function clears the trigger event on a digital I/O line. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].clear()
```

N	Digital I/O trigger line (1 to 14)

Details

The event detector of a trigger enters the detected state when an event is detected. It is cleared when `digio.trigger[N].wait()` or `digio.trigger[N].clear()` is called.

`digio.trigger[N].clear()` clears the event detector of the specified trigger line, discards the history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

`digio.trigger[2].clear()`	Clears the trigger event detector on I/O line 2.

Also see

digio.trigger[N].overrun (on page 8-124)
digio.trigger[N].wait() (on page 8-129)

digio.trigger[N].EVENT_ID

This constant identifies the trigger event generated by the digital I/O line N. This constant is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = digio.trigger[N].EVENT_ID
```

eventID	The trigger event number
N	Digital I/O trigger line (1 to 14)

Details

To have another trigger object respond to trigger events generated by the trigger line, set the other object's stimulus attribute to the value of this constant.

Example 1

`digio.trigger[5].stimulus = digio.trigger[3].EVENT_ID`	Uses a trigger event on digital I/O trigger line 3 to be the stimulus for digital I/O trigger line 5.

Example 2

`scan.trigger.arm.stimulus = digio.trigger[3].EVENT_ID`	Uses a trigger event on digital I/O trigger line 3 to be the stimulus for starting a scan.

Also see

None

digio.trigger[N].mode

This attribute sets the mode in which the trigger event detector and the output trigger generator operate on the given trigger line. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger *N* reset Recall setup	Create configuration script Save setup	0 (digio.TRIG_BYPASS)

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

triggerMode	The trigger mode; see **Details** for values
N	Digital I/O trigger line (1 to 14)

Details

Set *triggerMode* to one of the following values:

Trigger mode values

triggerMode	**Description**
`digio.TRIG_BYPASS` or `0`	Allows direct control of the line.
`digio.TRIG_FALLING` or `1`	Detects falling-edge triggers as input; asserts a TTL-low pulse for output.
`digio.TRIG_RISING` or `2`	If the programmed state of the line is high, the `digio.TRIG_RISING` mode behavior is similar to `digio.TRIG_RISINGA`. If the programmed state of the line is low, the `digio.TRIG_RISING` mode behavior is similar to `digio.TRIG_RISINGM`. This setting should only be used if necessary for compatibility with other Keithley Instruments products.
`digio.TRIG_EITHER` or `3`	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
`digio.TRIG_SYNCHRONOUSA` or `4`	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting the output trigger releases the latched line.
`digio.TRIG_SYNCHRONOUS` or `5`	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
`digio.TRIG_SYNCHRONOUSM` or `6`	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
`digio.TRIG_RISINGA` or `7`	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
`digio.TRIG_RISINGM` or `8`	Asserts a TTL-high pulse for output. Input edge detection is not possible in this mode.

When programmed to any mode except `digio.TRIG_BYPASS`, the output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

Use of either `digio.TRIG_SYNCHRONOUSA` or `digio.TRIG_SYNCHRONOUSM` is preferred over `digio.TRIG_SYNCHRONOUS`, because `digio.TRIG_SYNCHRONOUS` is provided for compatibility with the digital I/O and TSP-Link triggering on other Keithley Instruments products.

To control the line state, set the mode to `digio.TRIG_BYPASS` and use the `digio.writebit()` and `digio.writeport()` commands.

Example

`digio.trigger[4].mode = 2`	Sets the trigger mode for I/O line 4 to `digio.TRIG_RISING`.

Also see

digio.trigger[N].clear() (on page 8-122)
digio.trigger[N].reset() (on page 8-126)
digio.writebit() (on page 8-129)
digio.writeport() (on page 8-130)
Scanning and triggering (on page 3-1)

digio.trigger[N].overrun

Use this attribute to read the event detector overrun status. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Digital I/O trigger *N* clear Digital I/O trigger *N* reset Recall setup	Not saved	Not applicable

Usage

overrun = digio.trigger[*N*].overrun

overrun	Trigger overrun state (`true` or `false`)
N	Digital I/O trigger line (1 to 14)

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 = digio.trigger[1].overrun` `print(overrun)`	If there is no trigger overrun, the following text is output: `false`

Also see

digio.trigger[N].clear() (on page 8-122)
digio.trigger[N].reset() (on page 8-126)

digio.trigger[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger *N* reset Recall setup	Create configuration script Save setup	10e-6 (10 µs) digital I/O lines 1 through 9 20 µs digital I/O lines 10 through 14

Usage

```
width = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = width
```

width	The pulse width (seconds)
N	Digital I/O trigger line (1 to 14)

Details

Setting *width* to zero (0) seconds asserts the trigger indefinitely. To release the trigger line, use `digio.trigger[N].release()`.

Example

`digio.trigger[4].pulsewidth = 20e-6`	Sets the pulse width for trigger line 4 to 20 µs.

Also see

digio.trigger[N].assert() (on page 8-121)
digio.trigger[N].reset() (on page 8-126)
digio.trigger[N].release() (on page 8-125)

digio.trigger[N].release()

This function releases an indefinite length or latched trigger. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].release()
```

N	Digital I/O trigger line (1 to 14)

Details

Releases a trigger that was asserted with an indefinite pulse width time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Example

`digio.trigger[4].release()`	Releases digital I/O trigger line 4.

Also see

digio.trigger[N].assert() (on page 8-121)
digio.trigger[N].pulsewidth (on page 8-125)

digio.trigger[N].reset()

This function resets trigger values to their factory defaults. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].reset()
```

N	Digital I/O trigger line (1 to 14)

Details

This function resets the following attributes to factory default settings:

- `digio.trigger[N].mode`
- `digio.trigger[N].pulsewidth`
- `digio.trigger[N].stimulus`

It also clears `digio.trigger[N].overrun`.

Example

```
digio.trigger[3].mode = 2
digio.trigger[3].pulsewidth = 50e-6
digio.trigger[3].stimulus = digio.trigger[5].EVENT_ID
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
   digio.trigger[3].stimulus)
digio.trigger[3].reset()
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth,
   digio.trigger[3].stimulus)
```

Set the digital I/O trigger line 3 for a falling edge with a pulsewidth of 50 microseconds.
Use digital I/O line 5 to trigger the event on line 3.
Reset the line back to factory default values.
Output before reset:

```
2.00000e+00        5.00000e-05        5.00000e+00
```

Output after reset:

```
0.00000e+00        1.00000e-05        0.00000e+00
```

Also see

digio.trigger[N].mode (on page 8-123)
digio.trigger[N].overrun (on page 8-124)
digio.trigger[N].pulsewidth (on page 8-125)
digio.trigger[N].stimulus (on page 8-127)

digio.trigger[N].stimulus

This attribute selects the event that causes a trigger to be asserted on the digital output line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Digital I/O trigger *N* reset Save setup	Create configuration script Save setup	0

Usage

```
triggerStimulus = digio.trigger[N].stimulus
digio.trigger[N].stimulus = triggerStimulus
```

triggerStimulus	The event identifier for the triggering event
N	Digital I/O trigger line (1 to 14)

Details

Set this attribute to zero (0) to disable the automatic trigger output.

Do not use the stimulus attribute for generating output triggers under script control. Use `digio.trigger[N].assert()` instead.

The trigger stimulus for a digital I/O line may be set to one of the existing trigger event IDs, described in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID`	A channel trigger event starts the scan.
`digio.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID`	The trigger key on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID`	A delay expired.
`tsplink.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY`	Scan ready event.
`scan.trigger.EVENT_SCAN_START`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP`	Scan complete event.
`scan.trigger.EVENT_IDLE`	Idle event.
`schedule.alarm[N].EVENT_ID`	A scan starts when alarm *N* fires.

Example 1

`digio.trigger[3].stimulus = 0`	Clear the trigger stimulus of digital I/O line 3.

Example 2

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

Set the trigger stimulus of digital I/O line 3 to be the channel ready event during a scan.

Also see

digio.trigger[N].assert() (on page 8-121)
digio.trigger[N].clear() (on page 8-122)
digio.trigger[N].reset() (on page 8-126)

digio.trigger[N].wait()

This function waits for a trigger. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = digio.trigger[N].wait(timeout)
```

triggered	The value `true` if a trigger is detected, or `false` if no triggers are detected during the timeout period
N	Digital I/O trigger line (1 to 14)
timeout	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `digio.trigger[N].wait()` or `digio.trigger[N].clear()` was called, this function returns a value immediately. After waiting for a trigger with this function, the event detector is automatically reset and ready to detect the next trigger. This is true regardless of the number of events detected.

Example

`triggered = digio.trigger[4].wait(3)` `print(triggered)`	Waits up to three seconds for a trigger to be detected on trigger line 4, then outputs the results. Output if no trigger is detected: `false` Output if a trigger is detected: `true`

Also see

digio.trigger[N].clear() (on page 8-122)

digio.writebit()

This function sets a digital I/O line high or low. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writebit(N, data)
```

N	Digital I/O trigger line (1 to 14)
data	The value to write to the bit: • 0 (low) • Non-zero (high)

Details

If the output line is write-protected using the `digio.writeprotect` attribute, the command is ignored.

The `reset()` function does not affect the present state of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

The data must be zero (0) to clear the bit. Any value other than zero (0) sets the bit.

Example

`digio.writebit(4, 0)`	Sets digital I/O line 4 low (0).

Also see

digio.readbit() (on page 8-120)
digio.readport() (on page 8-120)
digio.trigger[N].mode (on page 8-123)
digio.writeport() (on page 8-130)
digio.writeprotect (on page 8-131)

digio.writeport()

This function writes to all digital I/O lines. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`digio.writeport(`*`data`*`)`

`data`	Value to write to the port (0 to 16383)

Details

The binary representation of *`data`* indicates the output pattern to be written to the I/O port. For example, a *`data`* value of 170 has a binary equivalent of 00000010101010. Lines 2, 4, 6, and 8 are set high (1), and the other 10 lines are set low (0).

Write-protected lines are not changed.

The `reset()` function does not affect the present states of the digital I/O lines.

Use the `digio.writebit()` and `digio.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `digio.TRIG_BYPASS`.

Example

`digio.writeport(255)`	Sets digital I/O Lines 1 through 8 high (binary 00000011111111).

Also see

digio.readbit() (on page 8-120)
digio.readport() (on page 8-120)
digio.writebit() (on page 8-129)
digio.writeprotect (on page 8-131)

digio.writeprotect

This attribute contains the write-protect mask that protects bits from changes from the `digio.writebit()` and `digio.writeport()` functions. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	0

Usage

```
mask = digio.writeprotect
digio.writeprotect = mask
```

mask	Sets the value that specifies the bit pattern for write-protect

Details

Bits that are set to one cause the corresponding line to be write-protected.

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a mask value of 7 has a binary equivalent of 00000000000111. This mask write-protects lines 1, 2, and 3.

Example

`digio.writeprotect = 15`	Write-protects lines 1, 2, 3, and 4.

Also see

digio.writebit() (on page 8-129)
digio.writeport() (on page 8-130)

display.clear()

This function clears all lines of the display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.clear()
```

Details

This function switches to the user screen and then clears the display.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the physical display as soon as processing time becomes available.

Also see

display.setcursor() (on page 8-145)
display.settext() (on page 8-146)

display.getannunciators()

This function reads the annunciators (indicators) that are presently turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`annunciators = display.getannunciators()`

annunciators	The bitmasked value that shows which indicators are turned on

Details

This function returns a bitmasked value showing which indicators are turned on. The 16-bit binary equivalent of the returned value is the bitmask. The return value is a sum of set annunciators, based on the weighted value, as shown in the following table.

Annunciator (indicator) bitmasked values and equivalent constants

Indicator	Bit	Weighted value	Equivalent constant
FILT	1	1	`display.ANNUNCIATOR_FILTER`
MATH	2	2	`display.ANNUNCIATOR_MATH`
4W	3	4	`display.ANNUNCIATOR_4_WIRE`
AUTO	4	8	`display.ANNUNCIATOR_AUTO`
ARM	5	16	`display.ANNUNCIATOR_ARM`
TRIG	6	32	`display.ANNUNCIATOR_TRIGGER`
* (star)	7	64	`display.ANNUNCIATOR_STAR`
SMPL	8	128	`display.ANNUNCIATOR_SAMPLE`
EDIT	9	256	`display.ANNUNCIATOR_EDIT`
ERR	10	512	`display.ANNUNCIATOR_ERROR`
REM	11	1024	`display.ANNUNCIATOR_REMOTE`
TALK	12	2048	`display.ANNUNCIATOR_TALK`
LSTN	13	4096	`display.ANNUNCIATOR_LISTEN`
SRQ	14	8192	`display.ANNUNCIATOR_SRQ`
REAR	15	16384	`display.ANNUNCIATOR_REAR`
REL	16	32768	`display.ANNUNCIATOR_REL`

Example 1

```
testAnnunciators = display.getannunciators()
print(testAnnunciators)

rem = bit.bitand(testAnnunciators, 1024)
if rem > 0 then
   print("REM is on")
else
   print("REM is off")
end
```

REM indicator is turned on.
Output:

```
1.28000e+03
REM is on
```

Example 2

`print(display.ANNUNCIATOR_EDIT)` `print(display.ANNUNCIATOR_TRIGGER)` `print(display.ANNUNCIATOR_AUTO)`	Output: `2.56000e+02` `3.20000e+01` `8.00000e+00`

Also see

bit.bitand() (on page 8-11)

display.getcursor()

This function reads the present position of the cursor on the front panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

row, *column*, *style* = `display.getcursor()`

row	The row where the cursor is: `1` (top row); `2` (bottom row)
column	The column where the cursor is: • If the cursor is in the top row: 1 to 20 • If the cursor is in the bottom row: 1 to 32
style	Visibility of the cursor: `0` (invisible cursor); `1` (blinking cursor)

Details

This function switches the display to the user screen (the text set by `display.settext()`), and then returns values to indicate the cursor's row and column position and cursor style.

Columns are numbered from left to right on the display.

Example 1

`testRow, testColumn = display.getcursor()` `print(testRow, testColumn)`	This example reads the cursor position into local variables and prints them. Example output: `1.00000e+00     1.00000e+00`

Example 2

`print(display.getcursor())`	This example prints the cursor position directly. In this example, the cursor is in row 1 at column 3, with an invisible cursor: `1.00000e+00   3.00000e+00` `0.00000e+00`

Also see

display.gettext() (on page 8-135)
display.screen (on page 8-143)
display.setcursor() (on page 8-145)
display.settext() (on page 8-146)

display.getlastkey()

This function retrieves the key code for the last pressed key.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
keyCode = display.getlastkey()
```

keyCode	A returned value that represents the last front-panel key pressed; see **Details** for more information

Details

A history of the key code for the last pressed front-panel key is maintained by the instrument. When the instrument is turned on, or when it is transitioning from local to remote operation, the key code is set to 0 (`display.KEY_NONE`).

Pressing the EXIT (LOCAL) key normally aborts a script. To use this function with the EXIT (LOCAL) key, you must set `display.locallockout` to `display.LOCK`.

The table below lists the *keyCode* value for each front-panel action.

Key codes			
Value	**Key list**	**Value**	**Key list**
0	display.KEY_NONE	82	display.KEY_ENTER
65	display.KEY_RANGEUP	83	display.KEY_REC
66	display.KEY_FUNC	84	display.KEY_DMM
67	display.KEY_REL	85	display.KEY_DELETE
68	display.KEY_MENU	86	display.KEY_STEP
69	display.KEY_CLOSE	87	display.KEY_CHAN
70	display.KEY_SLOT	90	display.KEY_RATE
71	display.KEY_RUN	91	display.KEY_LIMIT
72	display.KEY_DISPLAY	92	display.KEY_TRIG
73	display.KEY_AUTO	93	display.KEY_OPEN
74	display.KEY_FILTER	94	display.KEY_PATT
75	display.KEY_EXIT	95	display.KEY_LOAD
76	display.KEY_STORE	97	display.WHEEL_ENTER
77	display.KEY_SCAN	103	display.KEY_RIGHT
78	display.KEY_INSERT	104	display.KEY_LEFT
79	display.KEY_OPENALL	107	display.WHEEL_LEFT
80	display.KEY_CONFIG	114	display.WHEEL_RIGHT
81	display.KEY_RANGEDOWN		

Example

`key = display.getlastkey()` `print(key)`	On the front panel, press the **MENU** key and then send the code shown here. This retrieves the key code for the last pressed key. Output: `6.80000e+01`

Also see

display.locallockout (on page 8-140)
display.sendkey() (on page 8-144)

display.gettext()

This function reads the text displayed on the instrument front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

text	The returned value, which contains the text that is presently displayed
embellished	Indicates type of returned text: `false` (simple text); `true` (text with embedded character codes)
row	Selects the row from which to read the text: `1` (row 1); 2 (row 2). If *row* is not included, both rows of text are read
columnStart	Selects the first column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; if nothing is selected, 1 is used
columnEnd	Selects the last column from which to read text; for row 1, the valid column numbers are 1 to 20; for row 2, the valid column numbers are 1 to 32; the default is 20 for row 1, and 32 for row 2

Details

Using the command without any parameters returns both lines of the display.

The `$N` character code is included in the returned value to show where the top line ends and the bottom line begins. This is not affected by the value of *embellished*.

When *embellished* is set to `true`, all other character codes are returned along with the message. When *embellished* is set to `false`, only the message and the `$N` character code is returned. For information on the embedded character codes, see display.settext() (on page 8-146).

The display is not switched to the user screen (the screen set using `display.settext()`). Text will be read from the active screen.

Example 1

```
display.clear()
display.setcursor(1, 1)
display.settext("ABCDEFGHIJ$DKLMNOPQRST")
display.setcursor(2, 1)
display.settext("abcdefghijklm$Bnopqrstuvwxyz$F123456")
print(display.gettext())
print(display.gettext(true))
print(display.gettext(false, 2))
print(display.gettext(true, 2, 9))
print(display.gettext(false, 2, 9, 10))
```

This example shows how to retrieve the display text in multiple ways. The output is:

```
ABCDEFGHIJKLMNOPQRST$Nabcdefghijklmnopqrstuvwxyz123456
$RABCDEFGHIJ$DKLMNOPQRST$N$Rabcdefghijklm$Bnopqrstuvwxyz$F123456
abcdefghijklmnopqrstuvwxyz123456
$Rijklm$Bnopqrstuvwxyz$F123456
ij
```

Example 2

```
display.clear()
display.settext("User Screen")
text = display.gettext()
print(text)
```

This outputs all text in both lines of the display:

```
User Screen       $N
```

This indicates that the message “User Screen” is on the top line. The bottom line is blank.

Also see

display.clear() (on page 8-131)
display.getcursor() (on page 8-133)
display.setcursor() (on page 8-145)
display.settext() (on page 8-146)

display.inputvalue()

This function displays a formatted input field on the instrument display that the operator can edit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.inputvalue(format)
display.inputvalue(format, default)
display.inputvalue(format, default, minimum)
display.inputvalue(format, default, minimum, maximum)
```

format	A string that defines how the input field is formatted; see Details for more information
default	The default value for the input value
minimum	The minimum input value
maximum	The maximum input value

Details

The *`format`* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted. The *`format`* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially	0.00000e+0
+	Allows operators to enter positive or negative values; if the "+" sign is not included, the operator cannot enter a negative value	+0.00
0	Defines the digit positions for the value; you can use up to six zeros (0)	+00.0000e+00
.	Include to have a decimal point appear in the value	+0.00

The *`default`* parameter is the value shown when the value is first displayed.

The *`minimum`* and *`maximum`* parameters can be used to limit the values that can be entered. When + is not selected for *`format`*, the minimum limit must be more than or equal to zero (0). When limits are used, you cannot enter values above or below these limits.

The input value is limited to ±1e37.

Before calling `display.inputvalue()`, you should send a message prompt to the operator using `display.prompt()`. Make sure to position the cursor where the edit field should appear.

After this command is sent, script execution pauses until you enter a value and press the **ENTER** key.

For positive and negative entry (plus sign (+) used for the value field and/or the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel . Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero (0) value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the EXIT (LOCAL) key, the function returns `nil`.

Example

```
display.clear()
display.settext("Enter value between$N  -0.10 and 2.00:  ")
value = display.inputvalue("+0.00", 0.5, -0.1, 2.0)
print("Value entered = ", value)
```

Displays an editable field (+0.50) for operator input. The valid input range is -0.10 to +2.00, with a default of 0.50.

Output:

```
Value entered =    1.35000e+00
```

Also see

display.prompt() (on page 8-142)
display.setcursor() (on page 8-145)
display.settext() (on page 8-146)

display.loadmenu.add()

This function adds an entry to the USER TESTS menu, which can be accessed by pressing the **LOAD** key on the instrument front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.loadmenu.add(displayName, code)
display.loadmenu.add(displayName, code, memory)
```

displayName	The name that is added to the USER TESTS menu
code	The code that is run from the USER TESTS menu
memory	Determines if code is saved to nonvolatile memory: `0` or `display.DONT_SAVE`: Does not save the code to nonvolatile memory `1` or `display.SAVE`: Saves the code to nonvolatile memory (default)

Details

After adding code to the load menu, you can run it from the front panel by pressing the **LOAD** key, then selecting **USER TESTS** to select from the available code to load. Pressing the **RUN** key will then run the script.

You can add items in any order. They are always displayed in alphabetic order when the menu is selected.

Any Lua code can be can be included in the *code* parameter. If *memory* is set to `display.SAVE`, the entry (name and code) is saved in nonvolatile memory. Scripts, functions, and variables used in the code are not saved by `display.SAVE`. Functions and variables need to be saved with the code. If the code is not saved in nonvolatile memory, it will be lost when the Series 3700A is turned off. See **Example 2** below.

If you do not make a selection for *memory*, the code is automatically saved to nonvolatile memory.

Quick Tip

You can create a script that defines several functions, and then use the `display.loadmenu.add()` command to add items that call those individual functions. This allows the operator to run tests from the front panel.

Example 1

`display.loadmenu.add("Test9", "Test9()")`	Assume a user script named "Test9" has been loaded into the run-time environment. Adds the menu entry to the User menu to run the script after loading.

Example 2

`display.loadmenu.add(` `   "Test", "DUT1() beeper.beep(2, 500)",` `   display.SAVE)`	Assume a script with a function named "DUT1" has already been loaded into the instrument, and the script has NOT been saved in nonvolatile memory. Now assume you want to add a test named "Test" to the USER TESTS menu. You want the test to run the function named "DUT1" and sound the beeper. This example adds "Test" to the menu, defines the code, and then saves the `displayName` and code in nonvolatile memory. When "Test" is run from the front panel USER TESTS menu, the function named "DUT1" executes and the beeper beeps for two seconds. Now assume you turn off instrument power. Because the script was not saved in nonvolatile memory, the function named "DUT1" is lost when you turn the instrument on. When "Test" is again run from the front panel, an error is generated because DUT1 no longer exists in the instrument as a function.

Example 3

`display.loadmenu.add("Part1",` `   "testpart([[Part1]], 5.0)", display.SAVE)`	Adds an entry called "Part1" to the front panel "USER TESTS" load menu for the code `testpart([[Part1]], 5.0)`, and saves it in nonvolatile memory.

Also see

display.loadmenu.delete() (on page 8-140)

display.loadmenu.catalog()

This function creates an iterator for the user menu items accessed using the LOAD key on the instrument front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for displayName in display.loadmenu.catalog() do body end
for displayName, code in display.loadmenu.catalog() do body end
```

displayName	The name displayed in the menu
code	The code associated with the *displayName*
body	The body of the code to process the entries in the loop

Details

Each time through the loop, *displayName* and *code* will take on the values in the USER TESTS menu. The instrument goes through the list in random order.

Example

`for displayName, code in` `   display.loadmenu.catalog() do` `   print(displayName, code)` `end`	Output: `Test DUT1() beeper.beep(2, 500)` `Part1 testpart([[Part1]], 5.0)` `Test9 Test9()`

Also see

display.loadmenu.add() (on page 8-138)
display.loadmenu.delete() (on page 8-140)

display.loadmenu.delete()

This function removes an entry from the USER TESTS menu, which can be accessed using the **LOAD** key on the instrument front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`display.loadmenu.delete(`*`displayName`*`)`

`displayName`	The name to be deleted from the USER TESTS menu

Details

If you delete an entry from the USER TESTS menu, you can no longer run it by pressing the **LOAD** key.

Example

`display.loadmenu.delete("Test9")` `for displayName, code in` `   display.loadmenu.catalog() do` `   print(displayName, code)` `end`	Deletes the entry named "Test9" Output: `Test   DUT1() beeper.beep(2, 500)` `Part1   testpart([[Part1]], 5.0)`

Also see

display.loadmenu.add() (on page 8-138)
display.loadmenu.catalog() (on page 8-139)

display.locallockout

This attribute describes whether or not the EXIT (LOCAL) key on the instrument front panel is enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (display.UNLOCK)

Usage

`lockout`` = display.locallockout`
`display.locallockout = `*`lockout`*

`lockout`	`0` or `display.UNLOCK`: Unlocks EXIT (LOCAL) key `1` or `display.LOCK`: Locks out EXIT (LOCAL) key

Details

Set `display.locallockout` to `display.LOCK` to prevent the user from interrupting remote operation by pressing the EXIT (LOCAL) key.

Set this attribute to `display.UNLOCK` to allow the EXIT (LOCAL) key to interrupt script or remote operation.

Example

`display.locallockout = display.LOCK`	Disables the front-panel EXIT (LOCAL) key.

Also see

None

display.menu()

This function presents a menu on the front panel display.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`selection` `= display.menu(`*`name`*`,` *`items`*`)`

`selection`	Name of the variable that holds the selected menu item
`name`	Menu name to display on the top line
`items`	Menu items to display on the bottom line

Details

The menu consists of the menu name string on the top line, and a selectable list of items on the bottom line. The menu items must be a single string with each item separated by whitespace. The name for the top line is limited to 20 characters.

After sending this command, script execution pauses for the operator to select a menu item. An item is selected by rotating the navigation wheel to place the blinking cursor on the item, and then pressing the navigation wheel (or the ENTER key). When an item is selected, the text of that selection is returned.

Pressing the EXIT (LOCAL) key will not abort the script while the menu is displayed, but it will return `nil`. The script can be aborted by calling the exit function when `nil` is returned.

Example

`selection = display.menu("Menu", "Test1 Test2 Test3")` `print(selection)`	Displays a menu with three menu items. If the second menu item is selected, selection is given the value `Test2`. Output: `Test2`

Also see

None

display.prompt()

This function prompts the user to enter a parameter from the front panel of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.prompt(format, units, help)
display.prompt(format, units, help, default)
display.prompt(format, units, help, default, minimum)
display.prompt(format, units, help, default, minimum, maximum)
```

format	A string that defines how the input field is formatted; see **Details** for more information
units	Set the units text string for the top line (eight characters maximum); this indicates the units (for example, "V" or "A") for the value
help	Text string to display on the bottom line (32 characters maximum)
default	The value that is shown when the value is first displayed
minimum	The minimum input value that can be entered
maximum	The maximum input value that can be entered (must be more than minimum)

Details

This function creates an editable input field at the present cursor position, and an input prompt message on the bottom line. Example of a displayed input field and prompt:

```
0.00V
Input 0 to +2V
```

The *format* parameter uses zeros (0), the decimal point, polarity sign, and exponents to define how the input field is formatted.

The *format* parameter can include the options shown in the following table.

Option	Description	Examples
E	Include the E to display the value exponentially. Include a plus sign (+) for positive/negative exponent entry. Do not include the plus sign (+) to prevent negative value entry. 0 defines the digit positions for the exponent.	`0.00000E+0`
+	Allows operators to enter positive or negative values. If the plus sign (+) is not included, the operator cannot enter a negative value.	`+0.00`
0	Defines the digit positions for the value. You can use up to six zeros (0).	`+00.0000E+00`
.	The decimal point where needed for the value.	`+0.00`

The *minimum* and *maximum* parameters can be used to limit the values that can be entered. When a plus sign (+) is not selected for *format*, the minimum limit must be greater than or equal to zero (0). When limits are used, the operator cannot enter values above or below these limits.

The input value is limited to ±1e37.

After sending this command, script execution pauses for the operator to enter a value and press **ENTER**.

For positive and negative entry (plus sign (+) used for the value field and the exponent field), polarity of a nonzero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After executing this command and pressing the **EXIT (LOCAL)** key, the value returns `nil`.

Example

```
value = display.prompt("0.00", "V", "Input 0 to +2V", 0.5, 0, 2)
print(value)
```

The above command prompts the operator to enter a voltage value. The valid input range is 0 to +2.00, with a default of 0.50:

```
0.50V
Input 0 to +2V
```

If the operator enters 0.70, the output is:

```
7.00000e-01
```

Also see

display.inputvalue() (on page 8-136)

display.screen

This attribute contains the selected display screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	display.MAIN

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Saved setup	Models 2601A/2611A/2635A: 0 (display.SMUA) Models 2602A/2604A/2612A/2614A/2636A: 2 (display.SMUA_SMUB)

Usage

```
displayID = display.screen
display.screen = displayID
```

displayID	One of the following values: • 1 or `display.MAIN`: Displays the main screen • 2 or `display.USER`: Displays the user screen

Details

Setting this attribute selects the display screen for the front panel. This performs the same action as pressing the DISPLAY key on the front panel. The text for the display screen is set by `display.settext()`.

Read this attribute to determine which of the available display screens was last selected.

NOTE

This does not support the CLOSED CHANNELS option that is available from the **DISPLAY** key.

Example

`display.screen = display.USER`	Selects the user display.

Also see

display.settext() (on page 8-146)

display.sendkey()

This function sends a code that simulates the action of a front panel control.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.sendkey(keyCode)
```

keyCode	A parameter that specifies the key press to simulate; see **Details** for more information

Details

This command simulates the pressing of a front panel key or navigation wheel, or the turning the navigation wheel one click to the left or right.

Key codes

Value	Key list		Value	Key list
0	`display.KEY_NONE`		82	`display.KEY_ENTER`
65	`display.KEY_RANGEUP`		83	`display.KEY_REC`
66	`display.KEY_FUNC`		84	`display.KEY_DMM`
67	`display.KEY_REL`		85	`display.KEY_DELETE`
68	`display.KEY_MENU`		86	`display.KEY_STEP`
69	`display.KEY_CLOSE`		87	`display.KEY_CHAN`
70	`display.KEY_SLOT`		90	`display.KEY_RATE`
71	`display.KEY_RUN`		91	`display.KEY_LIMIT`
72	`display.KEY_DISPLAY`		92	`display.KEY_TRIG`
73	`display.KEY_AUTO`		93	`display.KEY_OPEN`
74	`display.KEY_FILTER`		94	`display.KEY_PATT`
75	`display.KEY_EXIT`		95	`display.KEY_LOAD`
76	`display.KEY_STORE`		97	`display.WHEEL_ENTER`
77	`display.KEY_SCAN`		103	`display.KEY_RIGHT`
78	`display.KEY_INSERT`		104	`display.KEY_LEFT`
79	`display.KEY_OPENALL`		107	`display.WHEEL_LEFT`
80	`display.KEY_CONFIG`		114	`display.WHEEL_RIGHT`
81	`display.KEY_RANGEDOWN`			

NOTE

When using this function, send built-in constants, such as `display.KEY_RIGHT`, rather than the numeric value, such as `103`. This allows for better forward compatibility with firmware revisions.

Example

`display.sendkey(display.KEY_RUN)`	Simulates pressing the RUN key.

Also see

Front panel

display.setcursor()

This function sets the position of the cursor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.setcursor(row, column)
display.setcursor(row, column, style)
```

row	The row number for the cursor (1 or 2)
column	The active column position to set; row 1 has columns 1 to 20, row 2 has columns 1 to 32
style	Set the cursor to invisible (0, default) or blinking (1)

Details

Sending this command selects the user screen and then moves the cursor to the given location.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the physical display as soon as processing time becomes available.

An out-of-range parameter for *row* sets the cursor to row 2. An out-of-range parameter for *column* sets the cursor to column 20 for row 1, or 32 for row 2.

An out-of-range parameter for *style* sets it to 0 (invisible).

A blinking cursor is only visible when it is positioned over displayed text. It cannot be seen when positioned over a space character.

Example

`display.clear()` `display.setcursor(1, 8)` `display.settext("Hello")` `display.setcursor(2, 14)` `display.settext("World")`	This example displays a message on the instrument front panel, approximately center. Note that the top line of text is larger than the bottom line of text. The front panel of the instrument displays "Hello" on the top line and "World" on the second line.

Also see

display.clear() (on page 8-131)
display.getcursor() (on page 8-133)
display.gettext() (on page 8-135)
display.screen (on page 8-143)
display.settext() (on page 8-146)

display.settext()

This function displays text on the user screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
display.settext(text)
```

text	Text message to be displayed, with optional character codes

Details

This function selects the user display screen and displays the given text.

After the instrument is turned on, the first time you use a display command to write to the display, the message "User Screen" is cleared. After the first write, you need to use `display.clear()` to clear the message.

The `display.clear()`, `display.setcursor()`, and `display.settext()` functions are overlapped commands. That is, the script does not wait for one of these commands to complete. These functions do not immediately update the display. For performance considerations, they update the physical display as soon as processing time becomes available.

The text starts at the present cursor position. After the text is displayed, the cursor is after the last character in the display message.

Top line text does not wrap to the bottom line of the display automatically. Any text that does not fit on the current line is truncated. If the text is truncated, the cursor remains at the end of the line.

The text remains on the display until replaced or cleared.

The character codes described in the following table can be also be included in the text string.

Display character codes

Character Code	Description
`$N`	Newline, starts text on the next line; if the cursor is already on line 2, text will be ignored after the `$N` is received
`$R`	Sets text to normal intensity, nonblinking
`$B`	Sets text to blink
`$D`	Sets text to dim intensity
`$F`	Sets the text to background blink
`$$`	Escape sequence to display a single dollar symbol ($)

Example

```
display.clear()
display.settext("Normal $BBlinking$N")
display.settext("$DDim $FBackgroundBlink$R $$$$ 2 dollars")
```

This example sets the display to:

```
Normal Blinking
Dim BackgroundBlink $$ 2 dollars
```

with the named effect on each word.

Also see

display.clear() (on page 8-131)
display.getcursor() (on page 8-133)
display.gettext() (on page 8-135)
display.screen (on page 8-143)
display.setcursor() (on page 8-145)

display.trigger.EVENT_ID

This constant is the event ID of the event generated when the front-panel TRIG key is pressed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

`eventID = display.trigger.EVENT_ID`

eventID	The trigger event number

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to front-panel trigger key events.

Example

```
scan.trigger.channel.stimulus = display.trigger.EVENT_ID
```
Have the channel action of the trigger model be paced by a user pressing the front-panel TRIG key.

Also see

scan.trigger.channel.stimulus (on page 8-341)

display.waitkey()

This function captures the key code value for the next front-panel action.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`keyCode = display.waitkey()`

keyCode	See **Details** for more information

Details

After you send this function, script execution pauses until a front-panel action (for example, pressing a key or the navigation wheel, or turning the navigation wheel). After the action, the value of the key (or action) is returned.

If the EXIT (LOCAL) key is pressed while this function is waiting for a front-panel action, the script is not aborted.

A typical use for this function is to prompt the user to press the EXIT (LOCAL) key to abort the script or press any other key to continue. For example, if the *keyCode* value 67 is returned (the EXIT (LOCAL) key was pressed), the `exit()` function can be called to abort the script.

The table below lists the *keyCode* value for each front panel action.

Key codes

Value	Key list	Value	Key list
0	display.KEY_NONE	83	display.KEY_RUN
66	display.KEY_DELETE	84	display.KEY_TRIG
67	display.KEY_EXIT	86	display.KEY_STEP
69	display.KEY_CLOSE	87	display.KEY_CHAN
70	display.KEY_SLOT	90	display.KEY_INSERT
72	display.KEY_DISPLAY	91	display.KEY_MENU
74	display.KEY_ENTER	93	display.KEY_OPEN
76	display.KEY_LOAD	94	display.KEY_PATT
77	display.KEY_SCAN	97	display.WHEEL_ENTER
79	display.KEY_OPENALL	107	display.WHEEL_LEFT
80	display.KEY_CONFIG	114	display.WHEEL_RIGHT

NOTE

When using this function, send built-in constants such as `display.KEY_STEP` (rather than the numeric value of 86). This will allow for better forward compatibility with firmware revisions.

Example

`key = display.waitkey()` `print(key)`	Pause script execution until the operator presses a key or the navigation wheel, or rotates the navigation wheel. If the output is: `8.600000000e+01` It indicates that the STEP key was pressed.

Also see

display.getlastkey() (on page 8-134)
display.sendkey() (on page 8-144)
display.settext() (on page 8-146)

dmm.adjustment.count

This attribute indicates the number of times the instrument has been adjusted (calibrated).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

```
calibrationCount = dmm.adjustment.count
```

calibrationCount	The number of times the instrument has been adjusted or calibrated

Details

Calibration (adjustment) may or may not be unlocked for this attribute to read and return a value.

Example

`adjustmentCount = dmm.adjustment.count`	Queries for the adjustment count.

Also see

dmm.adjustment.date (on page 8-149)
dmm.calibration.unlock() (on page 8-165)

dmm.adjustment.date

This attribute sets or queries the calibration adjustment date in Coordinated Universal Time (UTC) format (number of seconds since January 1, 1970).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	DMM nonvolatile memory	Not applicable

Usage

```
calibrationDate = dmm.adjustment.date
dmm.adjustment.date = os.time({year=yyyy, month=m, day=d})
```

calibrationDate	The number of seconds since January 1, 1970
os.time{year=*yyyy*, month=*m*, day=*d*}	Specifies the date; if a value is not specified, sets the adjustment date to the present date of the instrument

Details

This attribute can only be set when calibration is unlocked.

For more information about formatting options with `os.time` or `os.date`, see the Lua documentation (*http://www.lua.org*).

Example 1

```
dmm.adjustment.date = os.time{year=2007, month=7, day = 4}
```

Sets the adjustment date to July 4, 2007.

Example 2

```
print(os.date("%m/%d/%Y", dmm.adjustment.date))
```

Queries the date and formats the response as mm/dd/yyyy:
`07/04/2007`

Example 3

```
print(os.date("%x", dmm.adjustment.date))
```

Queries the date and formats the response as mm/dd/yy:
`02/24/09`

Also see

dmm.adjustment.count (on page 8-148)
dmm.calibration.unlock() (on page 8-165)

dmm.aperture

The aperture setting for the active DMM function in seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Function change DMM close Recall DMM configuration Instrument reset DMM reset Recall setup	Save DMM configuration Create configuration script Save setup	60 Hz: 1.666666667e-002 50 Hz: 2.000000000e-002

Usage

```
value = dmm.aperture
dmm.aperture = value
```

value	Represents the desired aperture: • For 50 Hz line frequency, the range is 10e-6 s to 0.250 s • For 60 Hz line frequency, the range is 8.33e-6 s to 0.250 s • For frequency and period, the range is 0.01 s to 0.273 s

Details

The `dmm.aperture` attribute is available for the following functions.

Function	Default value
"accurrent"	1.666667e-02
"acvolts"	1.666667e-02
"commonsideohms"	1.666667e-02
"dccurrent"	1.666667e-02
"dcvolts"	1.666667e-02
"fourwireohms"	1.666667e-02
"frequency"	1.000000e-02
"period"	1.000000e-02
"temperature"	1.666667e-02
"twowireohms"	1.666667e-02

The aperture setting is not available for the functions `"continuity"` and `"nofunction"`. If you query the aperture when either of these functions is selected, nil is returned. If you write the command when either of these functions is selected, an error is generated.

The aperture value is saved with the `dmm.func` function setting, so if you use another function, then return to the previous setting, such as `"dcvolts"` or `"frequency"`, the aperture value you set previously is retained.

The setting for aperture may be automatically adjusted based on what the DMM supports. Therefore, after setting the aperture, query the value to see if it was adjusted.

If the detector bandwidth (`dmm.detectorbandwidth`) setting is 30 or less for `"acvolts"` or `"accurrent"`, an error message is generated if you try to set the aperture for these functions.

Example

`dmm.func = "dcvolts"` `dmm.aperture = 16.67e-3`	Set the aperture to 16.67 milliseconds for DC volts.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.detectorbandwidth (on page 8-179)
dmm.func (on page 8-187)
dmm.nplc (on page 8-217)

dmm.appendbuffer()

Appends data from the reading buffer to a file on the USB flash drive. If no file exists, this function creates a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.appendbuffer(bufferVar, fileName)
dmm.appendbuffer(bufferVar, fileName, timeFormat)
```

bufferVar	A string with the name of a DMM reading buffer from which you want to append data to the specified file
fileName	A string with the file name of the file on the USB flash drive to which reading buffer data will be appended
timeFormat	How the date and time information should be saved. The values for *timeFormat* are: • `dmm.buffer.SAVE_RELATIVE_TIME`: Saves relative time stamps only • `dmm.buffer.SAVE_FORMAT_TIME`: Saves dates, times and fractional seconds. This is the default if no time format is specified • `dmm.buffer.SAVE_RAW_TIME`: Saves seconds and fractional seconds only • `dmm.buffer.SAVE_TIMESTAMP_TIME`: Saves only time stamps

Details

For options that save more than one item of time information, each item is comma delimited. For example, the default format will be `<date>`, `<time>`, and `<fractional seconds>` for each reading, separated by commas.

The file extension .csv is appended to the filename if necessary. Any file extension other than .csv generates errors.

Because `dmm.appendbuffer()` appends data, it does not include header information. The `dmm.savebuffer()` function does included header information.

The index column entry starts at 1 for each append operation, which is also what the `dmm.savebuffer()` command does.

> **NOTE**
>
> The reading buffer files saved to the USB flash drive will always have an extension of .csv.

Errors are generated:

- If the reading buffer does not exist.
- If the reading buffer is not a DMM buffer.
- If the destination filename is not specified correctly.
- If the file extension is not set to .csv. (You can leave the file extension blank.)

Examples of valid destination file names:

```
dmm.appendbuffer("bufferVar", "/usb1/myData")
dmm.appendbuffer("bufferVar", "/usb1/myData.csv")
```

Invalid destination filename examples:

```
dmm.appendbuffer("bufferVar", "/usb1/myData.")
```

— The period is not followed by the csv extension.

```
dmm.appendbuffer("bufferVar", "/usb1/myData.txt")
```

— The only allowed extension is .csv. If .csv is not assigned, it is automatically added.

```
dmm.appendbuffer("bufferVar", "/usb1/myData.txt.csv")
```

— Two periods in the file name (`myData_txt.csv` would be correct).

Example

`dmm.appendbuffer("bufferVar", "/usb1/myData.csv")`	Appends readings from a valid DMM buffer named bufferVar with default time information to a file named `myData.csv` on the USB flash drive.
`dmm.appendbuffer("bufferVar", "/usb1/myDataRel.csv", dmm.buffer.SAVE_RELATIVE_TIME)`	Appends readings from bufferVar with relative time stamps to a file named myDataRel.csv on the USB flash drive.

Also see

dmm.makebuffer() (on page 8-204)
dmm.savebuffer() (on page 8-236)

dmm.autodelay

The autodelay setting for the selected DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Function change DMM close Recall DMM configuration Reset DMM reset Recall setup	Create configuration script Save setup Save DMM configuration	2 (`dmm.AUTODELAY_ONCE`)

Usage

```
value = dmm.autodelay
dmm.autodelay = value
```

state	Enable autodelay (`dmm.ON` or `1`) Disable autodelay (`dmm.OFF` or `0`) Enable autodelay for the first measurement only (`dmm.AUTODELAY_ONCE` or 2)

Details

The autodelay setting applies to the function selected by `dmm.func`. It is available for all functions except `"nofunction"`.

To have the DMM include a delay before each measurement, set auto delay to `dmm.ON` or `1`.

To have the DMM take a measurement without an automatic delay, set auto delay to `dmm.OFF` or `0`.

To take a measurement for the first measurement in a set or group of measurements, you can use `dmm.AUTODELAY_ONCE` or 2. The delay occurs only on the first measurement of each set of measurements. If `dmm.measurecount` is set to `1`, `dmm.AUTODELAY_ONCE` acts similarly to On, applying a delay at the start of every measurement.

An error is generated if you try to set autodelay for `"nofunction"`. Error code `1114`, "Setting conflicts with function selected," is generated. If you query autodelay for `"nofunction"`, `nil` is returned with the same error.

Example

`dmm.func = "twowireohms"` `dmm.autodelay = dmm.ON` `dmm.measurecount = 10` `ReadingBufferOne = dmm.makebuffer(1000)` `dmm.measure(ReadingBufferOne)`	An automatic delay is applied to each measurement when the DMM is measuring two-wire ohms. Take 10 measurements and store them in a reading buffer named `ReadingBufferOne` that can store up to 1000 readings.
`dmm.func = "dcvolts"` `dmm.autodelay = dmm.OFF`	No delay is applied is applied to the DC volt measurements.
`dmm.func = "fourwireohms"` `dmm.autodelay = dmm.AUTODELAY_ONCE` `dmm.measurecount = 10` `ReadingBufferTwo = dmm.makebuffer(1000)` `dmm.measure(ReadingBufferTwo)`	Sets an auto delay for the first of the ten four-wire ohm readings. Readings two through ten will occur as quickly as possible, with readings stored in a reading buffer called ReadingBufferTwo that can store up to 1000 readings.

Also see

Autodelay (on page 4-5)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.measurecount (on page 8-214)

dmm.autorange

The auto range setting for the active DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.ON)

Usage

```
value = dmm.autorange
dmm.autorange = value
```

value	Auto range: • Enable: dmm.ON or 1 • Disable: dmm.OFF or 0

Details

Auto range selects the best range in which to measure the applied signal. The instrument will auto range at 100% of range. When auto range is enabled, upranging occurs at 120% of range and downranging occurs when the reading is <10% of nominal range. For example, if you are on the 10 volt range and auto range is enabled, the instrument will auto range up to the 100 volts range when the measurement exceeds 12 volts and will auto range down to the 1 volt range when the measurement falls below 1 volt.

The auto range setting applies to the function selected by `dmm.func`. Auto range is available for the following functions:

- `"accurrent"` or `dmm.AC_CURRENT`
- `"acvolts"` or `dmm.AC_VOLTS`
- `"commonsideohms"` or `dmm.COMMON_SIDE_OHMS`
- `"dccurrent"` or `dmm.DC_CURRENT`
- `"dcvolts"` or `dmm.DC_VOLTS`
- `"fourwireohms"` or `dmm.FOUR_WIRE_OHMS`
- `"twowireohms"` or `dmm.TWO_WIRE_OHMS`

Auto range is not available for any other functions. If you try to set auto range for any other function, an error is returned. If you query the auto range for any other function, nil is returned and an error is generated.

The auto range value is saved with the `dmm.func` function setting, so if you use another function, then return to the previous setting, such as `"dcvolts"` or `"fourwireohms"`, the autorange setting you set previously is retained. With auto range enabled, you can use the `dmm.range` command to view the range that is presently being used. Using `dmm.range` to select a fixed range disables auto range.

Example 1

`dmm.func = "twowireohms"` `dmm.autorange = dmm.ON`	Enable auto ranging for 2-wire ohms.

Example 2

`dmm.func = "dcvolts"`	Set DMM function to be DC volts.
`dmm.reset("active")`	Reset only the active DMM function (DC volts).
`print(dmm.autorange, dmm.range)`	View the default auto range and range selection.
`dmm.range = 50e-3`	Select a range suitable for a 50 mV reading.
`print(dmm.autorange, dmm.range)`	View the default auto range and range selection. Output: `1.000000000e+00    1.000000000e+01` `0.000000000e+00    1.000000000e-01`

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.range (on page 8-222)

dmm.autozero

The autozero setting for the active DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.ON)

Usage

```
value = dmm.autozero
dmm.autozero = value
```

value	Enable autozero (`dmm.ON` or `1`) Disable autozero (`dmm.OFF` or `0`) Refresh the reference points once then disable (`dmm.AUTOZERO_ONCE` or 2)

Details

The autozero setting applies to the function selected by `dmm.func`. It is available for all functions except `"continuity"` and `"nofunction"`.

You can send `dmm.AUTOZERO_ONCE` or 2 to refresh the reference points once. When this command is sent, the reference points are refreshed, and then autozero is set to disabled (`dmm.OFF` or `0`). Querying `dmm.autozero` after sending `dmm.AUTOZERO_ONCE` generates a response of `0`.

For `dmm.nplc` settings that are less than 0.2 plc, sending `dmm.AUTOZERO_ONCE` or 2 results in significant delays. For example, the delay time at a NPLC of 0.0005 is 2.75 s. The delay time at 0.199 is 5.45 s.

An error is generated if:

- You try to set `dmm.autozero` for `"continuity"` or `"nofunction"`. Error code 1114, "Setting conflicts with function selected," is generated.
- You query `dmm.autozero` for `"continuity"` or `"nofunction"`. `nil` is returned with error code 1114, "Setting conflicts with function selected."

Example

`dmm.func = "dcvolts"` `dmm.autozero = dmm.ON`	Enables autozero for DC volts.
`dmm.autozero = dmm.AUTOZERO_ONCE` `print(dmm.autozero)`	Refreshes the reference points once and sets autozero to `dmm.OFF` or `0`. Output: `0.000000000e+00`
`timer.reset()` `dmm.autozero=2` `time=timer.measure.t()` `print(time)`	Determines the time delay when autozero is selected.

Also see

Autozero
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.nplc (on page 8-217)
dmm.reset() (on page 8-228)
reset() (on page 8-317)

dmm.buffer.catalog()

Creates an iterator for the user-created reading buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in dmm.buffer.catalog() do...end
```

name	A string representing the name of a user created DMM reading buffer

Details

You can access the catalog for the user-created local reading buffers so that you can print the names of all reading buffers in the system. The entries are enumerated in no particular order. From this list, you may selectively delete reading buffers from the system.

NOTE

Do not delete the reading buffers by sending:

```
for name in dmm.buffer.catalog() do name = nil end
```

This locks the system and forces you to stop the command (through the EXIT key on the front panel). It does not delete the reading buffers from the instrument. This occurs because *name* is a string type variable and not a reading buffer type.

Example

`for name in dmm.buffer.catalog() do print(name) end`	Print all user-created local reading buffers in the system. Assume the return is: `buf3` `buf5` `buf1`
`buf1 = nil` `collectgarbage()`	Deletes `buf1`.

Also see

dmm.buffer.info() (on page 8-159)

dmm.buffer.info()

Returns the size and capacity of the reading buffer parameter.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

size, *capacity* = dmm.buffer.info(*bufferVar*)

size	Number representing the *N* (presently stored) attribute of the reading buffer parameter
capacity	Number representing the overall capacity attribute of the reading buffer parameter
bufferVar	String representing the reading buffer name that you want to query for size and capacity

Details

This function uses the specified reading buffer input parameter name to find the corresponding size and capacity to return. Use this function with the `dmm.buffer.catalog()` function to output the size and capacity for all reading buffers in the system.

Example

```
for n in dmm.buffer.catalog() do
   print(dmm.buffer.info(n))
end
```

Assume the system has the following reading buffers created: buffer1, buffer2, buffer3, buffer4, and buffer5.
Query the system for the size and capacity of each reading buffer without formatting the results.
The output is:

```
0.000000000e+00 2.000000000e+03
0.000000000e+00 4.000000000e+03
0.000000000e+00 5.000000000e+03
0.000000000e+00 3.000000000e+03
0.000000000e+00 1.000000000e+03
```

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

Query the system for the name, size, and capacity of each reading buffer while formatting the results.
The output is:

```
buffer2 size = 0 capacity = 2000
buffer4 size = 0 capacity = 4000
buffer5 size = 0 capacity = 5000
buffer3 size = 0 capacity = 3000
buffer1 size = 0 capacity = 1000
```

Also see

dmm.buffer.catalog() (on page 8-158)

dmm.buffer.maxcapacity

The overall maximum capacity for reading buffers in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Never	Not applicable	Not applicable

Usage

maximumCapacity = dmm.buffer.maxcapacity

maximumCapacity	A number that represents the overall maximum capacity for the reading buffers

Details

Determines the maximum capacity of the instrument for reading buffer storage. This value represents the total system reading buffer storage size. A single reading buffer may be created with this as its size, or several reading buffers may be created in the instrument that are smaller in size. However, the sum total of all reading buffer sizes in the instrument cannot exceed this maximum.

Example

`MaxBuffCap = dmm.buffer.maxcapacity` `print(MaxBuffCap)`	Reads the maximum reading buffer capacity for the instrument, which is 650,000 readings. Output: `6.500000000e+05`

Also see

dmm.buffer.info() (on page 8-159)
dmm.buffer.usedcapacity (on page 8-160)
dmm.makebuffer() (on page 8-204)

dmm.buffer.usedcapacity

Indicates how much of the maximum capacity for reading buffers in the instrument is used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup	Not applicable	Not applicable

Usage

usedCapacity = dmm.buffer.usedcapacity

usedCapacity	The presently used capacity for reading buffers in the instrument

Details

This value represents the sum total capacity of all reading buffers in the instrument.

Example

`buf1 = dmm.makebuffer(300000)` `buf2 = dmm.makebuffer(300000)` `print(dmm.buffer.usedcapacity)` `print(dmm.buffer.maxcapacity -` `   dmm.buffer.usedcapacity)`	Create buffers. Reads the used reading buffer capacity for the system. 6.000000000e+05 5.000000000e+04 This shows that there is 50,000 available for creating additional reading buffers.

Also see

dmm.buffer.info() (on page 8-159)
dmm.buffer.maxcapacity (on page 8-160)

dmm.calibration.ac()

Begins the desired AC adjustment step on the DMM.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.calibration.ac(step)
dmm.calibration.ac(step, value)
```

step	The AC adjustment step to perform
value	The value for this adjustment step (if the adjustment step has a value)

Details

This command generates an error if the:

- Calibration is locked
- Step is out of sequence
- Step does not exist
- Step does not complete successfully
- Value passed is invalid for the step, out of range, or not needed

Example

For detail on how to use `dmm.calibration.ac()`, see AC volts calibration (on page C-29), AC current calibration (on page C-31), and Frequency calibration (on page C-33).

Also see

dmm.calibration.dc() (on page 8-162)
dmm.calibration.lock() (on page 8-163)
dmm.calibration.unlock() (on page 8-165)

dmm.calibration.dc()

Begins the desired DC adjustment step on the DMM.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.calibration.dc(step)
dmm.calibration.dc(step, value)
```

step	The DC adjustment step to perform
value	The value for this adjustment step (if the adjustment step has a value)

Details

This command generates an error if the:

- Calibration is locked
- Step is out of sequence
- Step does not exist
- Step does not complete successfully
- Value passed is invalid for the step, out of range, or not needed

Example

For example of use, see DC volts calibration (on page C-25), Resistance calibration (on page C-27), and DC current calibration (on page C-28).

Also see

dmm.calibration.ac() (on page 8-161)
dmm.calibration.lock() (on page 8-163)
dmm.calibration.unlock() (on page 8-165)

dmm.calibration.lock()

Locks calibration to prevent unintended changes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.calibration.lock()
```

Details

Use this command to lock an unlocked calibration. An error is generated if this command is issued when calibration is already locked.

Once locked, you must unlock calibration before you can perform calibration again.

NOTE

Save calibration data before locking. Calibration data will be lost if it is not saved before locking.

Example

`dmm.calibration.save()` `dmm.calibration.lock()`	Save calibration, then lock it.

Also see

dmm.calibration.unlock() (on page 8-165)
dmm.calibration.save() (on page 8-165)

dmm.calibration.password

This attribute sets the password that must be entered before you can unlock calibration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	Not applicable	DMM nonvolatile memory	KI003706

Usage

```
dmm.calibration.password = password
```

password	A string that represents the valid password to unlock calibration

Details

This attribute can only be set when calibration is unlocked.

This attribute generates an error if calibration is locked or if the password string length is greater than ten characters.

NOTE

Be sure to record the password; there is no command to query for the password once it is set.

Example

`dmm.calibration.unlock("KI003706")` `dmm.calibration.password = "myUnlock"` `dmm.calibration.lock()`	To change the default calibration password, unlock the calibration with the default password. Saves the password as "myUnlock". Lock calibration. Subsequent unlocks will use the password "myUnlock".

Also see

dmm.calibration.unlock() (on page 8-165)

dmm.calibration.save()

Saves calibration data.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.calibration.save()
```

Details

This command saves the calibration constants and adjustment date and increases the adjustment count by 1. The adjustment count is the number of times calibration has been saved.

This command does not check for errors in calibration data. Calibration data is saved regardless of calibration data errors.

The calibration date can be specified with `dmm.adjustment.date`. If it is not specified, the date is based on the system date.

To prevent data loss, you need to send the save command before locking calibration.

An error is generated if this command is issued when calibration is already locked.

Example

`dmm.calibration.save()`	Saves calibration data.

Also see

dmm.adjustment.date (on page 8-149)
dmm.calibration.lock() (on page 8-163)
dmm.calibration.unlock() (on page 8-165)

dmm.calibration.unlock()

Unlocks calibration if calibration was locked.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.calibration.unlock(password)
```

password	A string representing the password to unlock calibration

Details

If the password does not match the saved password, an error is generated. The default password from the factory is "KI003706". You can change the default with `dmm.calibration.password`.

Example

`dmm.calibration.unlock("KI003706")`	Unlocks calibration using the default password.

Also see

dmm.calibration.lock() (on page 8-163)
dmm.calibration.password (on page 8-164)

dmm.calibration.verifydate

This attribute sets or queries the calibration verification date in UTC format (number of seconds since January 1, 1970).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	DMM nonvolatile memory	Not applicable

Usage

```
calibrationVerificationDate = dmm.calibration.verifydate
dmm.calibration.verifydate = os.time()
```

calibrationVerificationDate	The number of seconds since January 1, 1970 when the last calibration verify date was set
os.time({year=yyyy, month=m, day=d})	Specifies the date; if a value is not specified, sets the verification date to the present date of the instrument

Details

When using the `os.time()` function:

- If no parameters are specified, the current date and time of the instrument is used. See example 4 below.
- Use a table with entries for year as *yyyy*, month as *mm* and day as *dd* to specify a date. See example 3 below.

See Lua documentation (*http://www.lua.org*) for the formatting options that are available for `os.date`.

This command can only be set when calibration is unlocked.

Example 1

`print(os.date("%m/%d/%Y", dmm.calibration.verifydate))`	Assume the system date is July 4, 2007 for this example; queries the calibration verification date and formats the response as mm/dd/yyyy: `07/04/2007`

Example 2

`print(os.date("%x", dmm.calibration.verifydate))`	Assume the system date is July 4, 2007 for this example; queries the date and formats the response as mm/dd/yy: `07/04/07`

Example 3

`dmm.calibration.verifydate = os.time{year=2007, month=7, day = 4}`	Set the calibration verification date to July 4, 2007.

Example 4

`dmm.calibration.verifydate = os.time()`	Set the calibration verification to the present date of the instrument.

Also see

dmm.adjustment.date (on page 8-149)
dmm.calibration.unlock() (on page 8-165)
Lua documentation (*http://www.lua.org*)

dmm.close()

Closes the specified channel or channel pattern to prepare for a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`dmm.close(`*`channelList`*`)`

`channelList`	A string listing the channel or channel pattern to close

Details

When you close a channel or channel pattern:

- Channels on all slots are opened if they interfere with measurement, including analog backplane relays 1 and 2 and commonside ohm backplane relays. The opening and closing of channels mimics that of `channel.exclusiveslotclose()`. Therefore, when using a for-loop with `dmm.close()` command, the last channel on each slot is closed at the end of the for loop execution.
- To have additional analog backplane relays 3 through 6 close, use them on an alternate slot. If they need to be on same slot, create a channel pattern.
- To have additional channels close, use patterns. When you use patterns, you must specify all items to close, including analog backplane relays. With patterns, there is no auto manipulation of analog backplane relays 1 and 2 as there is with channels.
- Any amp channels will open. If you need to have multiple amp channels closed, create a channel pattern.
- Associated channels and analog backplane relays will close, which include analog backplane relay 1 and 2, as needed based on configuration associated with channel (see `dmm.getconfig()`) Analog backplane relays specified by `channel.setbackplane()` are not used.

The DMM configuration is determined by the configuration associated with the channel or channel pattern being closed. If the configuration is a default name, the function of that configuration will be reset to factory default settings. You must create a unique DMM configuration to avoid using factory default settings when assigning to a channel. For more information on setting DMM configuration, see dmm.configure.set() (on page 8-175), dmm.setconfig() (on page 8-237), and dmm.getconfig() (on page 8-189).

This command allows you to separate the closing of channels from measuring. Therefore, you may execute any number of commands between the close and measure commands to satisfy your application needs.

An error is generated if:

- The specified channel or channel pattern is invalid.
- The channel number does not exist for the slot specified.
- The channel pattern does not exist.
- The specified channel does not support being closed (like a digital I/O channel).
- More than one channel or channel pattern is specified.
- The channel is paired with another bank for a multi-wire application.
- The channel is an analog backplane relay.
- The channel configuration is set to nofunction.

Once an error is detected, the command stops processing. Channels and DMM settings remain unchanged.

Example

`dmm.setconfig("3003", "tempMeasure")` `dmm.close('3003')`	Close channel 3 on slot 3 and prepare the DMM for measuring temperature with `tempMeasure` settings.
`dmm.setconfig("channelDCV", "dcvolts")` `dmm.close("channelDCV")`	Close a channel pattern called `channelDCV` and prepare DMM for measuring DC volts at factory default settings.

Also see

channel.exclusiveslotclose() (on page 8-57)
channel.getclose() (on page 8-61)
channel.getstate() (on page 8-75)
channel.setbackplane() (on page 8-90)
dmm.getconfig() (on page 8-189)
dmm.open() (on page 8-219)

dmm.configure.catalog()

Creates an iterator for user-created DMM configurations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in dmm.configure.catalog() do … end
```

name	A string representing the name of a user-created DMM configuration.

Details

You can access the catalog for user DMM configurations to print or delete all configurations in the run-time environment.

The entries are enumerated in no particular order. This only lists user-created DMM configurations; it does not list the factory default configurations.

Example

`for name in dmm.configure.catalog() do` ` print(name)` `end`	Prints the names of all user-created DMM configurations in the instrument. The output will look similar to: `TestDcv` `TestTemperature` `TestTwoWire` This indicates there are three user-created DMM configurations in the instrument with the names TestDCV, TestTemperature, and TestTwoWire.
`for name in dmm.configure.catalog() do` ` dmm.configure.delete(name)` `end`	Deletes all user-created DMM configurations from the instrument.

Also see

dmm.configure.delete() (on page 8-170)
dmm.configure.query() (on page 8-171)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)

dmm.configure.delete()

Deletes a user-created DMM configuration from memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.configure.delete(name)
```

name	String that contains the name of the DMM configuration to delete

Details

If you delete a DMM configuration that is assigned to channels or channel patterns, those channels and patterns revert back to the factory default DMM configuration of `"nofunction"` (`dmm.setconfig()`).

If you delete a DMM configuration that is used in a scan list, the scan list is modified and the channel is set to `"nofunction"` for that configuration.

You cannot delete a DMM configuration on a closed channel. If you attempt to delete it, error code 1114, "Settings conflict with deleting DMM configuration assigned to closed channel," is generated.

An error is generated if the name specified does not exist as a user configuration.

Example

`dmm.configure.delete("DCVDMMConfig")`	Deletes a user configuration called `DCVDMMConfig`.

Also see

dmm.configure.catalog() (on page 8-169)
dmm.configure.set() (on page 8-175)
dmm.configure.query() (on page 8-171)
dmm.configure.recall() (on page 8-173)

dmm.configure.query()

Lists DMM settings associated with a configuration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
config = dmm.configure.query(userConfiguration)
config = dmm.configure.query(userConfiguration, userSeparator)
```

config	An output string that represents the DMM attribute settings of *userConfiguration*
userConfiguration	A string that contains the name for the DMM configuration to be listed. To query the settings for the active function, set this parameter to "active"
userSeparator	A string that represents the two-character separator that is inserted between items.The default value is a comma followed by a space (,)

Details

For the specified configuration, this function lists the settings and the corresponding DMM attributes.

If the specified configuration does not exist, a nil response is generated, along with an error message stating that the referenced name does not exist.

If *userSeparator* is specified, the attributes are delimited by this two-character separator. If more than two characters are specified, an error message is generated.

To query the factory default settings for a function, use the function, such as `"dvolts"` or `"temperature"`, for the *userConfiguration* parameter. See `dmm.func` for valid functions.

Example

```
dmm.configure.set("DCvConfig")
DCvConfigItems = dmm.configure.query("DCvConfig")
print(DCvConfigItems)
```

Creates the configuration DCvConfig. Lists the DMM attributes in DCvConfig, separated by commas.
Output:

```
function = dcvolts,nplc = 5.000000E-001,aperture = 8.333333E-003,range =
    1.000000E+001,auto zero = 0,auto delay = 2,filter enable = 0,filter type =
    1,filter count =  10,filter window = 1.000000E-001,rel enable = 0,rel level =
    0.000000E+000,display digits =  6,dB reference = 1.000000E+000,input divider =
    0,units = 0,line sync = 0,limit 1 enable = 1,limit 1 autoclear = 1,limit 1 low
    value = -3.000000E+000,limit 1 high value = 5.000000E+000,limit 2 enable =
    0,limit 2 autoclear = 1,limit 2 low value = -2.000000E+000,limit 2 high value =
    2.000000E+000,math enable = 0,math format = 2,math mxb mfactor =
    1.000000E+000,math mxb bfactor = 0.000000E+000,math mxb units = X,math percent
    = 1.000000E+000
```

```
DCvConfigItems = dmm.configure.query("DCvConfig", "\n")
print(DCvConfigItems)
```

Lists the DMM attributes in DCvConfig separated by new lines. Output: `function = dcvolts` `nplc = 5.000000E-001` `aperture = 8.333333E-003` `range = 1.000000E+001` `auto zero = 0` `auto delay = 2` `filter enable = 0` `filter type = 1` `filter count =  10` `filter window = 1.000000E-001` `rel enable = 0` `rel level = 0.000000E+000` `display digits =  6` `dB reference = 1.000000E+000` `input divider = 0` `units = 0` `line sync = 0` `limit 1 enable = 1` `limit 1 autoclear = 1` `limit 1 low value = -3.000000E+000` `limit 1 high value = 5.000000E+000` `limit 2 enable = 0` `limit 2 autoclear = 1` `limit 2 low value = -2.000000E+000` `limit 2 high value = 2.000000E+000` `math enable = 0` `math format = 2` `math mxb mfactor = 1.000000E+000` `math mxb bfactor = 0.000000E+000` `math mxb units = X` `math percent = 1.000000E+000`
`FactoryDCV = dmm.configure.query("dcvolts", "\n")` `print(FactoryDCV)`
Lists the factory default settings for DC volts separated by new lines.
`ActiveFunc = dmm.configure.query("active", "\n")` `print(ActiveFunc)`
Lists the DMM attributes for the active function separated by new lines.

Also see

dmm.configure.catalog() (on page 8-169)
dmm.configure.delete() (on page 8-170)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.configure.recall()

Recalls a user or factory DMM configuration and replaces attributes in the present configuration with attributes from the recalled version.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.configure.recall(configuration)
```

configuration	A string that represents the name of the DMM configuration to recall

Details

This command recalls the DMM configuration for one function.

When a configuration is recalled, the function associated with the configuration becomes active.

When you recall a DMM configuration, the existing DMM configuration settings for the function are replaced by the settings in the recalled configuration. Settings for other functions are not affected. For example, if the function associated with the configuration was temperature, only temperature settings are recalled. If a factory configuration is recalled, the function's attributes are set to their factory default values.

The DMM configuration can be user-defined or factory-defined.

User-defined DMM configurations are set with `dmm.configure.set()`. The factory-defined DMM configurations are:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"continuity"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"frequency"`
- `"nofunction"`
- `"period"`
- `"temperature"`
- `"twowireohms"`

Example

```
dmm.func = "dcvolts"
dmm.reset("active")
dmm.nplc = 0.5
dmm.range = 10
dmm.configure.set("TestDcv")
dmm.configure.recall("dcvolts")
print(dmm.func, dmm.autorange, dmm.range, dmm.nplc)
dmm.configure.recall("TestDcv")
print(dmm.func, dmm.autorange, dmm.range, dmm.nplc)
dmm.setconfig("slot1", "TestDcv")
dmm.setconfig("2001:2015", "TestDcv")
dmm.setconfig("3005", "TestDcv")
```

Set the DMM to the DC volts function.
Reset DC volts back to factory defaults.
Set the NPLC for DC volts to 0.5.
Select the 10 volt range for DC volts and disable autorange.
Save a user DMM configuration for DC volts called "TestDcv".
Recall and configure the DMM for factory DC volts.
Output the settings for factory-defined DC volts.
Recall the user DMM configuration called "TestDcv".
Output the settings for TestDcv.
Set the DMM configuration for slot 1, channels 2001 to 2005, and channel 3005 to TestDcv.
Output:

```
dcvolts     1.000000000e+00     1.000000000e+01     1.000000000e+00
dcvolts     0.000000000e+00     1.000000000e+01     5.000000000e-01
```

Also see

dmm.configure.delete() (on page 8-170)
dmm.configure.query() (on page 8-171)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.configure.set()

Creates a named DMM configuration for the selected function. The configuration includes pertinent attributes for that function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Reset Recall setup	Create configuration script Save setup	See **Details**

Usage

`dmm.configure.set(`*`name`*`)`

`name`	A string that contains the name of the DMM configuration that you are creating

Details

This command saves the selected function and its pertinent settings. You can recall this configuration using `dmm.configure.recall()`. You can also apply the configuration using `dmm.setconfig()` to channels or channel patterns.

`dmm.configure.set()` stores only pertinent settings. For example, if `dmm.func` is set to `"dcvolts"`, temperature settings are not stored.

DMM configurations are not saved through a power cycle. To save the configuration through a power cycle, use `setup.save()` or `createconfigscript()`. These options save all DMM user configurations.

If the name of the configuration:

- Already exists, the existing configuration is overwritten with the new configuration.
- Is the same as that of a factory-default configuration, an error is generated.
- Is longer than 30 characters, an error is generated.
- Any channels that were configured to use that configuration will be evaluated to determine if the new settings are valid for the channels. If they are, the channels will start using the new configuration settings. If not, the configuration associated with that channel will revert to the factory default setting of `"nofunction"`.

Some DMM configurations are preset. The factory default configuration names are:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"continuity"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"frequency"`
- `"nofunction"`
- `"period"`
- `"temperature"`
- `"twowireohms"`

If you change the settings for an existing DMM configuration, the existing scan list will be updated to use the new settings for the existing DMM configuration. However, if the function or a setting is not valid for a channel in the scan list, the scan list will be reset to the default configuration of `"nofunction"`.

Example

```
dmm.func = "dcvolts"
dmm.reset("active")
dmm.nplc = 0.5
dmm.range = 10
dmm.configure.set("TestDcv")
dmm.configure.recall("dcvolts")
print(dmm.func, dmm.autorange, dmm.range, dmm.nplc)
dmm.configure.recall("TestDcv")
print(dmm.func, dmm.autorange, dmm.range, dmm.nplc)
```

Set the DMM to the DC volts function.
Reset DC volts back to factory defaults.
Set the NPLC for DC volts to 0.5.
Select the 10 volt range for DC volts and disable autorange.
Save a user DMM configuration for DC volts called "TestDcv".
Recall and configure the DMM for factory DC volts.
Output the settings for factory DC volts.
Recall the user DMM configuration called "TestDcv".
Output the settings for the TestDcv configuration.
Output:

```
dcvolts     1.000000000e+00     1.000000000e+01     1.000000000e+00
dcvolts     0.000000000e+00     1.000000000e+01     5.000000000e-01
```

Also see

createconfigscript() (on page 8-115)
dmm.configure.catalog() (on page 8-169)
dmm.configure.delete() (on page 8-170)
dmm.configure.query() (on page 8-171)
dmm.configure.recall() (on page 8-173)
dmm.func (on page 8-187)
dmm.setconfig() (on page 8-237)
setup.save() (on page 8-370)

dmm.connect

Indicates how the DMM relays should be connected to the analog backplane.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	7 (dmm.CONNECT_ALL)

Usage

```
value = dmm.connect
dmm.connect = value
```

value	The DMM relay connection setting. See **Details** for valid values

Details

NOTE

Use of this command is not recommended with the exception of special cases. The default setting should handle most applications.

This setting affects all DMM functions. Valid values are shown in the table below.

Valid values	
Value	**Relays connected**
dmm.CONNECT_NONE or 0	None
dmm.CONNECT_ALL or 7	All
dmm.CONNECT_TWO_WIRE or 1	2-wire
dmm.CONNECT_FOUR_WIRE or 3	2-wire and sense
dmm.CONNECT_TWO_WIRE_AMPS or 5	2-wire & amp
dmm.CONNECT_AMPS or 4	amp

The relays are bitmapped into the lower 3 bits of the value as shown in the following table.

Relay bitmap		
Bit	**Value**	**Relays represented**
0	1	2-wire
1	2	sense
2	4	amp

To close a relay, set the appropriate bit to 1.
To open a relay, set the appropriate bit to 0.
An error is generated if:

- The sense relay bit is set to a 1 and the sense relay with amps is selected. These two settings correspond to a value of 2 or 6, respectively.
- The value is less than 0 or greater than 7.

Example

`dmm.connect = dmm.CONNECT_TWO_WIRE_AMPS`	Connects the DMM 2-wire and amp relays to the analog backplane.

Also see

None

dmm.dbreference

The decibel (DB) reference setting for the DMM in volts.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+00

Usage

```
value = dmm.dbreference
dmm.dbreference = value
```

value	The desired DB reference in volts (1e-7 to 1000)

Details

The DB reference setting applies only when `dmm.func` is set to `"dcvolts"` or `"acvolts"`. If you query this value for any other function, nil is returned.

An error is generated:

- If you send this command for any function other than `"dcvolts"` or `"acvolts"`.
- If the value is out of range.

The DB reference setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"dcvolts"` or `"acvolts"`, the DB reference setting you set previously are retained.

Example

`dmm.func = "dcvolts"` `dmm.dbreference = 5`	Sets the DB reference to 5 volts for DC volts.

Also see

Express DC or AC voltage in decibels (see "dB commands" on page 4-48)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.detectorbandwidth

The AC detector bandwidth setting for the DMM in Hertz.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	300

Usage

```
value = dmm.detectorbandwidth
dmm.detectorbandwidth = value
```

value	The detector bandwidth in Hertz

Details

Only applies when `dmm.func` is set to `"acvolts"` or `"accurrent"`.

If you query this value for any other function, nil is returned.

When you send this value, the input value is adjusted as follows:

Write value	Read value
< 30	3
Between 30 and 300	30
≥ 300	300

An error is generated:

- If you send this command for any function other than `"accurrent"` or `"acvolts"`
- If you set `dmm.aperture` and the detector bandwidth read value is 30 or less
- If the value is below 3

Example

```
reset()
dmm.func = "acvolts"
print(dmm.func, dmm.detectorbandwidth)
dmm.detectorbandwidth = 35
print(dmm.func, dmm.detectorbandwidth)
dmm.func = "accurrent"
print(dmm.func, dmm.detectorbandwidth)
dmm.func = "acvolts"
print(dmm.func, dmm.detectorbandwidth)
```

Sets the detector bandwidth to 35 Hz for AC volts. 35 is adjusted to 30. AC current is still at 300 Hz.

Output:

```
acvolts      3.000000000e+02
acvolts      3.000000000e+01
accurrent    3.000000000e+02
acvolts      3.000000000e+01
```

Also see

dmm.aperture (on page 8-150)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.displaydigits

The display digits setting for the selected DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	See table in **Details**

Usage

```
value = dmm.displaydigits
dmm.displaydigits = value
```

value	7½ display digits: `dmm.DIGITS_7_5` or 7 6½ display digits: `dmm.DIGITS_6_5` or 6 5½ display digits: `dmm.DIGITS_5_5` or 5 4½ display digits: `dmm.DIGITS_4_5` or 4 3½ display digits: `dmm.DIGITS_3_5` or 3

Details

This is not available for `"nofunction"`.

This attribute affects how the reading for a function 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 `dmm.func` function setting, so if you use another function, then return to the function for which you set display digits, the display digits setting you set previously is retained.

To change the number of digits returned in a remote command reading, use `format.asciiprecision`.

Defaults	
If `dmm.func` is…	**The default is…**
"accurrent", "acvolts", "temperature"	5
"commonsideohms", "dccurrent", "dcvolts", "twowireohms", "fourwireohms", "frequency", "period"	6
"continuity"	4

An error is generated:

- If the value is invalid
- If `dmm.func` is set to "nofunction", if the command is queried, `nil` is returned
- `dmm.func` is set to "nofunction" or "continuity", if the command is written, error code 1114, "Settings conflict with function selected," is returned

Example

`dmm.func = "dcvolts"` `dmm.displaydigits = dmm.DIGITS_7_5`	Enables display digits to 7½ for DC volts.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
format.asciiprecision (on page 8-255)

dmm.drycircuit

The dry circuit setting for the selected DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
state = dmm.drycircuit
dmm.drycircuit = state
```

state	Enable dry circuit (`dmm.ON` or `1`) Disable dry circuit (`dmm.OFF` or `0`)

Details

The dry circuit setting only applies when `dmm.func` is set to `"fourwireohms"` or `"commonsideohms"`.

For power and low-glitch resistance measurements requiring a low open-circuit voltage (20 mV), dry circuit ohms can be used on the 1 Ω, 10 Ω, 100 Ω, 1 kΩ, and 10 kΩ ranges (maximum of 2.4 kΩ) for the 4-wire ohm function. When dry circuit is enabled, offset compensation is automatically set to on.

This command automatically sets `dmm.offsetcompensation` to `dmm.ON` when `dmm.func` = `"fourwireohms"` or `"commonsideohms"`.

An error is generated if:

- You try to set `dmm.drycircuit` for a function other than `"fourwireohms"` or `"commonsideohms"`. Error 1114, "Setting conflicts with function selected" is generated.
- You query `dmm.drycircuit` for a function other than `"fourwireohms"` or `"commonsideohms"`. `nil` is returned, along with error 1114.
- The state is invalid.

The dry circuit setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the dry circuit setting you set previously is retained.

Example

`dmm.func = "fourwireohms"` `dmm.drycircuit = dmm.ON`	Enable dry circuit for 4-wire ohms.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.offsetcompensation (on page 8-218)

dmm.filter.count

The filter count setting for the selected DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	10

Usage

```
value = dmm.filter.count
dmm.filter.count = value
```

value	The filter count setting from 1 to 100

Details

The number of measured readings that will yield one filtered measurement when filtered measurements are enabled.

The filter count setting only applies when `dmm.func` is set to one of the following:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- "temperature"
- `"twowireohms"`

If you query the setting for any other function, `nil` is returned.

The filter count setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the filter count setting you set previously is retained.

An error is generated if:

- You send the setting for any other function.
- The value is out of range.

Example

`dmm.func = "twowireohms"` `dmm.filter.count = 5` `dmm.filter.enable = dmm.ON`	Sets the filter count for 2-wire ohms to 5 and enables filtered measurements.

Also see

dmm.configure.set() (on page 8-175)
dmm.configure.recall() (on page 8-173)
dmm.filter.enable (on page 8-183)
dmm.filter.type (on page 8-184)

dmm.filter.enable

Indicates if filtered measurements are enabled or disabled for the selected DMM function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
value = dmm.filter.enable
dmm.filter.enable = value
```

value	Filter measurements setting: • dmm.ON or 1: Filter measurements enabled • dmm.OFF or 0: Filter measurements disabled

Details

The filter enable setting only applies when `dmm.func` is set to one of the following:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"temperature"`
- `"twowireohms"`

Querying the setting for any other function will return `nil` and an error message.

The filter enable setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the filter enable setting you set previously is retained.

Example

`dmm.func = "twowireohms"` `dmm.filter.type = dmm.FILTER_MOVING_AVG` `dmm.filter.count = 3` `dmm.filter.enable = dmm.ON`	Enable filtered measurements for 2-wire ohms using a moving average filter type with a count of 3 for each measurement.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.filter.count (on page 8-182)
dmm.filter.type (on page 8-184)
dmm.filter.window (on page 8-185)
dmm.func (on page 8-187)
dmm.reset() (on page 8-228)

dmm.filter.type

The filter type for the DMM measurements for selected DMM functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.FILTER_REPEAT_AVG)

Usage

```
value = dmm.filter.type
dmm.filter.type = value
```

value	The filter type setting: • dmm.FILTER_MOVING_AVG or 0 for moving average filter • dmm.FILTER_REPEAT_AVG or 1 for repeating filter

Details

The filter type setting only applies when `dmm.func` is set to one of the following:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"temperature"`
- `"twowireohms"`

Querying the setting for any other function returns `nil`. An error is generated if this setting is written or read for any other function.

You can choose from two averaging filter types: Repeating and moving. When the repeating filter type is selected, the stack (filter count) is filled, and the conversions are averaged to yield a reading. The stack is then cleared, and the process starts over.

When the moving average filter type is selected, a first-in, first-out stack is used. When the stack (filter count) becomes full, the measurement conversions are averaged to yield a reading. For each subsequent conversion placed into the stack, the oldest conversion is discarded. The stack is then re-averaged to yield a new reading.

The filter type setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the filter type setting you set previously is retained.

Example

`dmm.func = "twowireohms"` `dmm.filter.type = dmm.FILTER_MOVING_AVG` `dmm.filter.enable = dmm.ON`	Set the filter type for 2-wire ohms to moving average and enable filtered measurements.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.filter.count (on page 8-182)
dmm.filter.enable (on page 8-183)
dmm.filter.window (on page 8-185)
dmm.func (on page 8-187)

dmm.filter.window

The filter window for the DMM measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E-01 (0.1)

Usage

```
value = dmm.filter.window
dmm.filter.window = value
```

value	The filter window setting; the range is between 0 and 10 to indicate percent of range

Details

The filter window setting only applies when `dmm.func` is set to one of the following:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"temperature"`
- `"twowireohms"`

Querying the setting for any other function returns `nil`. An error is generated if this setting is written or read for any other function.

An error is generated if the value is out of range.

The filter window setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the filter window setting you set previously is retained.

Example

`dmm.func = "twowireohms"` `dmm.filter.window = 0.25` `dmm.filter.enable = dmm.ON`	Set the filter window for 2-wire ohms to 0.25 and enable filtered measurements.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.filter.enable (on page 8-183)
dmm.filter.count (on page 8-182)
dmm.filter.type (on page 8-184)

dmm.fourrtd

The type of four-wire RTD being used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.RTD_PT100)

Usage

```
value = dmm.fourrtd
dmm.fourrtd = value
```

value	The desired type for four-wire RTD: • dmm.RTD_PT100 or 0 for type PT100 • dmm.RTD_D100 or 1 for type D100 • dmm.RTD_F100 or 2 for type F100 • dmm.RTD_PT385 or 3 for type PT385 • dmm.RTD_PT3916 or 4 for type PT3916 • dmm.RTD_USER or 5 for user-specified type

Details

This attribute is only valid when `dmm.func` is set to `"temperature"` and `dmm.transducer` is set to `dmm.TEMP_FOURRTD`. For all other transducer types, the attribute is set but is not used until the transducer type is set for four-wire RTD.

All other functions generate an error and return nil when queried. An illegal parameter value error message is generated if the value specified is not a supported RTD type value as listed in the usage table.

The four-wire RTD setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the four-wire RTD settings you set previously are retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.fourrtd = dmm.RTD_PT3916`	Sets the type of four-wire RTD for PT3916.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)

dmm.func

The selected function for the DMM.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset DMM reset Recall setup	Create configuration script Save setup	dmm.DC_VOLTS

Usage

```
function = dmm.func
dmm.func = function
```

function	One of the following DMM functions:	
	• "accurrent" or dmm.AC_CURRENT • "acvolts" or dmm.AC_VOLTS • "commonsideohms" or dmm.COMMON_SIDE_OHMS • "continuity" or dmm.CONTINUITY • "dccurrent" or dmm.DC_CURRENT • "dcvolts" or dmm.DC_VOLTS	• "fourwireohms" or dmm.FOUR_WIRE_OHMS • "frequency" or dmm.FREQUENCY • "nofunction" or dmm.NO_FUNCTION • "period" or dmm.PERIOD • "temperature" or dmm.TEMPERATURE • "twowireohms" or dmm.TWO_WIRE_OHMS

Details

This attribute determines the selected DMM function and indicates how the other DMM attributes are to be processed.

When the DMM functionality changes, the attributes for the new DMM function become active. Unless you update these attributes, they will be the factory defaults or the values that were used the last time the function was used. If you want to see settings for a particular function, change to that function with `dmm.func`, then write or read the desired settings specifically. To see all attributes at once, use `dmm.configure.query` with a first parameter value of "active" as shown in the example below.

An error is generated:

- If the setting does not match one of the ones specified in usage.
- If a user DMM configuration name is used to set the function.

If an error is found, no change is made to the function.

Example

`dmm.func = "temperature"`	Makes "temperature" the active DMM function.
`dmm.func = "dcvolts"` `dcv_nplc = dmm.nplc`	Check the NPLC setting for DC volts.
`dmm.func = dmm.DC_VOLTS` `dmm.nplc = 0.5` `dmm.range = 10` `dmm.func = "twowireohms"` `dmm.nplc = 0.1` `dmm.range = 100000` `dmm.func = "dcvolts"` `print(dmm.nplc)` `print(dmm.range)` `dmm.func = dmm.TWO_WIRE_OHMS` `print(dmm.nplc)` `print(dmm.range)`	Example showing how the instrument retains values for each function. Output: `0.5` `10` `.1` `100000`

`dmm.func = "dcvolts"` `print(dmm.configure.query("active",` `"\n"))`	Select DC volts for the DMM function, then query the active settings to see how the DC volts function is presently configured. Example output: `function = dcvolts` `nplc = 5.000000E-001` `aperture = 8.333333E-003` `range = 1.000000E+001` `auto zero = 1` `auto delay = 2` `filter enable = 0` `filter type = 1` `filter count =   3` `filter window = 5.300000E+000` `rel enable = 0` `rel level = 0.000000E+000` `display digits =  6` `dB reference = 1.000000E+000` `input divider = 0` `units = 0` `line sync = 0` `limit 1 enable = 0` `limit 1 autoclear = 1` `limit 1 low value = -1.000000E+000` `limit 1 high value = 1.000000E+000` `limit 2 enable = 0` `limit 2 autoclear = 1` `limit 2 low value = -2.000000E+000` `limit 2 high value = 2.000000E+000` `math enable = 0` `math format = 2` `math mxb mfactor = 1.000000E+000` `math mxb bfactor = 0.000000E+000` `math mxb units = X` `math percent = 1.000000E+000`

Also see

dmm.configure.query() (on page 8-171)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)

dmm.getconfig()

Queries for the DMM configurations that are associated with the specified channels or channel patterns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Recall setup	Create configuration script Save setup	"nofunction"

Usage

DMMconfiguration = dmm.getconfig(*channelList*)

DMMconfiguration	A comma-delimited string that lists the DMM configurations associated with items in channelList
channelList	The channels or channel patterns to query

Details

The response is a comma-delimited string that lists the user-defined and factory-defined configurations. They are listed in the same order in which they are specified in channelList.

The configurations indicate how the DMM will be configured when the corresponding channel or channel pattern is closed with the dmm.close() function or used in a scan list without an overriding DMM configuration.

An error is generated if:

- A specified channel or channel pattern is invalid.
- A channel number does not exist for slot based on installed card.
- Channel pattern does not exist.
- Channel being specified does not support a configuration setting (for example, a digital I/O channel or analog backplane relay).

Example

slot1_2Configs = dmm.getconfig("slot1, slot2") print(slot1_2Configs)	Queries channels on slots 1 and 2.
chan3001_3010Configs = dmm.getconfig("3001:3010") print(chan3001_3010Configs)	Queries channels 1 to 10 on slot 3. Sample output may be: dcvolts,dcvolts,dcvolts,dcvolts,dcvolts,temperature,temperature,temperature,temperature,temperature This shows that channels 3001 to 3005 are configured for "dcvolts" and 3006 to 3010 are configured for "temperature".

Also see

dmm.close() (on page 8-167)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.setconfig() (on page 8-237)
scan.add() (on page 8-319)
scan.create() (on page 8-325)

dmm.inputdivider

Enables or disables the 10 M ohm input divider.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
state = dmm.inputdivider
dmm.inputdivider = state
```

`state`	Enable input divider (`dmm.ON` or `1`) Disable input divider (`dmm.OFF` or `0`)

Details

This attribute is only valid when `dmm.func` is set to DC volts.

The input divider setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"dcvolts"`, the input divider setting you set previously is retained.

An error is generated if you try to set input divider for any DMM function other than "dcvolts". Error code 1114, "Setting conflicts with function selected," is generated. If you query any DMM function other than "dcvolts" for input divider, `nil` is returned with the same error.

Example

`dmm.func = "dcvolts"` `dmm.inputdivider = dmm.ON`	Enables the input divider for DC volts.

Also see

DC volts input divider (on page 4-57)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.limit[Y].autoclear

Indicates if limit *Y* should be cleared automatically or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.ON)

Usage

```
value = dmm.limit[Y].autoclear
dmm.limit[Y].autoclear = value
```

value	The auto clear setting: • Enable: dmm.ON or 1 • Disable: dmm.OFF or 0
Y	1 or 2 for limit number

Details

This attribute is valid for all functions except `"continuity"` and `"nofunction"`. A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

When this attribute is enabled, a limit fail condition tracks how the measurements are taken. If a measurement fails limit, the fail indication is set. If the next measurement passes limit, the failed limit condition clears. Therefore, if you are scanning or taking a series of measurements with auto clear enabled for a limit, the last measurement limit dictates the fail indication for the limit.

To know if any of a series of measurements failed the limit, set the auto clear setting to off. When set to `dmm.OFF`, a failed indication will not be cleared automatically and will remain set until it is cleared by `dmm.limit[Y].clear()`.

The auto clear setting affects both the high and low limits of *Y*.

Example

`dmm.func = "twowireohms"` `dmm.limit[2].autoclear = dmm.ON`	Enables auto clear on limit 2 for two-wire ohms.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].clear() (on page 8-192)
dmm.measure() (on page 8-213)

dmm.limit[Y].clear()

Clears the test results of limit *Y*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.limit[Y].clear()
```

Y	1 or 2 for limit number

Details

Use this command to clear the test results of limit *Y* when the limit auto clear (`dmm.limit[Y].autoclear`) command is disabled. Both the high and low test results are cleared.

To avoid the need to manually clear the test results for a limit, enable the auto clear command.

Example

`dmm.func = "twowireohms"` `dmm.limit[2].clear()`	Clears the test results for the high and low limit 2 for two-wire ohms.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.limit[Y].autoclear (on page 8-191)
dmm.limit[Y].high.fail (on page 8-195)
dmm.limit[Y].low.fail (on page 8-199)

dmm.limit[Y].enable

Enable or disable limit *Y* testing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
value = dmm.limit[Y].enable
dmm.limit[Y].enable = value
```

value	Limit *Y* testing: • Enable: dmm.ON or 1 • Disabled: dmm.OFF or 0
Y	1 or 2 for limit number

Details

This attribute is valid for all functions except "continuity" and "nofunction". A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

When this attribute is enabled, the limit *Y* testing occurs on each measurement taken by the DMM. Limit *Y* testing compares the measurements to the high and low limit values. If a measurement falls outside these limits, the test fails. The low limit is specified by `dmm.limit[Y].low.value` and the high limit is specified by `dmm.limit[Y].high.value`.

When this is enabled, limit testing occurs whether it is requested by the `dmm.measure` function or as part of a scan sequence. However, if events are not assigned to a trigger stimulus for a digital I/O line, there is no hardware indication of limits. The events that can be assigned to a trigger stimulus include:

- `dmm.trigger.EVENT_LIMIT1_HIGH`
- `dmm.trigger.EVENT_LIMIT1_LOW`
- `dmm.trigger.EVENT_LIMIT2_HIGH`
- `dmm.trigger.EVENT_LIMIT2_LOW`

To see the test results, use the `dmm.limit[Y].low.fail` and `dmm.limit[Y].high.fail` attributes.

When limit testing is disabled, no measurements are tested and the status bits are not updated, the fail indication does not get updated, and hardware lines are not generated.

Example

This example enables limits 1 and 2 for DC volt, measurements. Limit 1 is checking for readings to be between 3 and 5 volts while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail will be 1 afterwards. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
dmm.func = "dcvolts"                -- set the DMM for DC volts functionality
dmm.reset("active")                 -- reset DC volts to default settings
dmm.range = 10                      -- set the range to 10 volts
dmm.nplc = 0.1                      -- set the nplc to 0.1
dmm.limit[1].autoclear = dmm.OFF    -- disable auto clearing for limit 1
dmm.limit[1].high.value = 5         -- set high limit on 1 to fail if reading
                                       -- exceeds 5 volts
dmm.limit[1].low.value = 3          -- set low limit on 1 to fail if reading
                                       -- is less than 3 volts
dmm.limit[1].enable = dmm.ON        -- enable limit 1 checking for DC volt
                                       -- measurements
dmm.limit[2].autoclear = dmm.OFF    -- disable auto clearing for limit 2
dmm.limit[2].high.value = 7         -- set high limit on 2 to fail if reading
                                       -- exceeds 7 volts
dmm.limit[2].low.value = 1          -- set low limit on 2 to fail if reading
                                       -- is less than 1 volts
dmm.limit[2].enable = dmm.ON        -- enable limit 2 checking for DC volt
                                       -- measurements
dmm.measurecount = 50               -- set the measure count to 50
LimitBuffer = dmm.makebuffer(100)   -- create a reading buffer that can store
                                       -- 100 readings
dmm.measure(LimitBuffer)            -- take 50 readings and store them in
                                       -- LimitBuffer
                                    -- then check if any of the 50 readings
                                       -- were outside of the limits
print("limit 1 high fail = " .. dmm.limit[1].high.fail)
print("limit 1 low fail = " .. dmm.limit[1].low.fail)
print("limit 2 high fail = " .. dmm.limit[2].high.fail)
print("limit 2 low fail = " .. dmm.limit[2].low.fail)
dmm.limit[1].clear()                -- clear limit 1 conditions
dmm.limit[2].clear()                -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 high fail = 0
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 high fail = 1
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit limit 1 and 2 low values (a 0.5 volts reading would cause this condition or a reading less than 1):

```
limit 1 high fail = 0
limit 1 low fail = 1
limit 2 high fail = 0
limit 2 low fail = 1
```

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].high.fail (on page 8-195)

dmm.limit[Y].high.value (on page 8-197)
dmm.limit[Y].low.fail (on page 8-199)
dmm.limit[Y].low.value (on page 8-201)
dmm.measure() (on page 8-213)

dmm.limit[Y].high.fail

Query for the high test results of limit *Y*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0

Usage

```
value = dmm.limit[Y].high.fail
```

value	The high fail indication for limit *Y*: • 0 indicates test passed – measurement within the high limit • 1 indicates test failed – measurement has exceeded high limit
Y	1 or 2 for limit number

Details

This attribute is valid for all functions except `"continuity"` and `"nofunction"`. A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

This attribute returns the results of high limit *Y* testing. If this is 1 (failed), the measurement was above the high limit (`dmm.limit[Y].high.value`).

Note that if you are scanning or taking a series of measurements with auto clear (`dmm.limit[Y].autoclear`) enabled for a limit, the last measurement limit dictates the fail indication for the limit. If autoclear is disabled, you can take a series of readings and read fails to see if any of one of the readings failed.

To use this attribute, you must set the limit to enable.

If autoclear and limit are not set, the high fail value indicates the results of the last limit test that occurred when limits were enabled.

In addition to this attribute, you can see the fail indication by reading the measurement event register of the status model. If the readings are stored in a reading buffer, the values are associated with *bufferVar*`.statuses` for the readings.

You can use the digital I/O line trigger stimulus commands to generate a pulse when a limit fails. The events that can be assigned to a trigger stimulus include:

- `dmm.trigger.EVENT_LIMIT1_HIGH`
- `dmm.trigger.EVENT_LIMIT1_LOW`
- `dmm.trigger.EVENT_LIMIT2_HIGH`
- `dmm.trigger.EVENT_LIMIT2_LOW`

Example

This example enables limits 1 and 2 for DC volt, measurements. Limit 1 is checking for readings to be between 3 and 5 volts while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail will be 1 afterwards. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
dmm.func = "dcvolts"                 -- set the DMM for DC volts functionality
dmm.reset("active")                  -- reset DC volts to default settings
dmm.range = 10                       -- set the range to 10 volts
dmm.nplc = 0.1                       -- set the nplc to 0.1
dmm.limit[1].autoclear = dmm.OFF     -- disable auto clearing for limit 1
dmm.limit[1].high.value = 5          -- set high limit on 1 to fail if reading
                                        -- exceeds 5 volts
dmm.limit[1].low.value = 3           -- set low limit on 1 to fail if reading
                                        -- is less than 3 volts
dmm.limit[1].enable = dmm.ON         -- enable limit 1 checking for DC volt
                                        -- measurements
dmm.limit[2].autoclear = dmm.OFF     -- disable auto clearing for limit 2
dmm.limit[2].high.value = 7          -- set high limit on 2 to fail if reading
                                        -- exceeds 7 volts
dmm.limit[2].low.value = 1           -- set low limit on 2 to fail if reading
                                        -- is less than 1 volts
dmm.limit[2].enable = dmm.ON         -- enable limit 2 checking for DC volt
                                        -- measurements
dmm.measurecount = 50                -- set the measure count to 50
LimitBuffer = dmm.makebuffer(100)    -- create a reading buffer that can store
                                        -- 100 readings
dmm.measure(LimitBuffer)             -- take 50 readings and store them in
                                        -- LimitBuffer
                                     -- then check if any of the 50 readings
                                        -- were outside of the limits
print("limit 1 high fail = " .. dmm.limit[1].high.fail)
print("limit 1 low fail = " .. dmm.limit[1].low.fail)
print("limit 2 high fail = " .. dmm.limit[2].high.fail)
print("limit 2 low fail = " .. dmm.limit[2].low.fail)
dmm.limit[1].clear()                 -- clear limit 1 conditions
dmm.limit[2].clear()                 -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 high fail = 0
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 high fail = 1
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit limit 1 and 2 low values (a 0.5 volts reading would cause this condition or a reading less than 1):

```
limit 1 high fail = 0
limit 1 low fail = 1
limit 2 high fail = 0
limit 2 low fail = 1
```

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].autoclear (on page 8-191)

dmm.limit[Y].high.value

The high limit value for limit *Y* when `dmm.limit[Y].enable` is set to `dmm.ON`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	limit 1: 1.000000E+00 limit 2: 2.000000E+00

Usage

```
value = dmm.limit[Y].high.value
dmm.limit[Y].high.value = value
```

value	The high value for limit *Y*; range is -4294967295 to +4294967295
Y	1 or 2 for limit number

Details

This attribute is valid for all functions except `"continuity"` and `"nofunction"`. A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

This attribute specifies or queries the high limit value of limit Y. When limit Y testing is enabled (`dmm.limit[Y].enable = 1`), a fail indication occurs when the measurement value is greater than this value.

You may set or get the value regardless if the limit is set to a digio trigger stimulus:

- `dmm.trigger.EVENT_LIMIT1_HIGH`
- `dmm.trigger.EVENT_LIMIT1_LOW`
- `dmm.trigger.EVENT_LIMIT2_HIGH`
- `dmm.trigger.EVENT_LIMIT2_LOW`

Example

This example enables limits 1 and 2 for DC volt, measurements. Limit 1 is checking for readings to be between 3 and 5 volts while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail will be 1 afterwards. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
dmm.func = "dcvolts"                 -- set the DMM for DC volts functionality
dmm.reset("active")                  -- reset DC volts to default settings
dmm.range = 10                       -- set the range to 10 volts
dmm.nplc = 0.1                       -- set the nplc to 0.1
dmm.limit[1].autoclear = dmm.OFF     -- disable auto clearing for limit 1
dmm.limit[1].high.value = 5          -- set high limit on 1 to fail if reading
                                        -- exceeds 5 volts
dmm.limit[1].low.value = 3           -- set low limit on 1 to fail if reading
                                        -- is less than 3 volts
dmm.limit[1].enable = dmm.ON         -- enable limit 1 checking for DC volt
                                        -- measurements
dmm.limit[2].autoclear = dmm.OFF     -- disable auto clearing for limit 2
dmm.limit[2].high.value = 7          -- set high limit on 2 to fail if reading
                                        -- exceeds 7 volts
dmm.limit[2].low.value = 1           -- set low limit on 2 to fail if reading
                                        -- is less than 1 volts
dmm.limit[2].enable = dmm.ON         -- enable limit 2 checking for DC volt
                                        -- measurements
dmm.measurecount = 50                -- set the measure count to 50
LimitBuffer = dmm.makebuffer(100)    -- create a reading buffer that can store
                                        -- 100 readings
dmm.measure(LimitBuffer)             -- take 50 readings and store them in
                                        -- LimitBuffer then check if any
                                     -- of the 50 readings were
                                        -- outside of the limits
print("limit 1 high fail = " .. dmm.limit[1].high.fail)
print("limit 1 low fail = " .. dmm.limit[1].low.fail)
print("limit 2 high fail = " .. dmm.limit[2].high.fail)
print("limit 2 low fail = " .. dmm.limit[2].low.fail)
dmm.limit[1].clear()                 -- clear limit 1 conditions
dmm.limit[2].clear()                 -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 high fail = 0
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 high fail = 1
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit limit 1 and 2 low values (a 0.5 volts reading would cause this condition or a reading less than 1):

```
limit 1 high fail = 0
limit 1 low fail = 1
limit 2 high fail = 0
limit 2 low fail = 1
```

Also see

Reading buffers (on page 3-55, on page 3-49)
digio.trigger[N].stimulus (on page 8-127)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].enable (on page 8-193)
dmm.limit[Y].high.fail (on page 8-195)
dmm.limit[Y].low.value (on page 8-201)

dmm.limit[Y].low.fail

Queries for the low test results of limit *Y*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0

Usage

```
value = dmm.limit[Y].low.fail
```

value	The low fail indication of limit *Y*: • Test passed: 0 (measurement above the low limit) • Test failed: 1 (measurement below the low limit)
Y	1 or 2 for limit number

Details

This attribute is valid for all functions except `"continuity"` and `"nofunction"`. A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

This attribute returns the results of low limit *Y* testing. If this is 1 (failed) is returned, the measurement was below the low limit.

Note that if you are scanning or taking a series of measurements with auto clear (`dmm.limit[Y].autoclear`) enabled for a limit, the last measurement limit dictates the fail indication for the limit. If autoclear is disabled, you can take a series of readings and read fails to see if any of one of the readings failed.

To use this attribute, you must set the limit to enable.

If autoclear and limit are not set, the low fail value indicates the results of the last limit test that occurred when limits were enabled.

In addition to this attribute, you can see the fail indication by reading the measurement event register of the status model. If the readings are stored in a reading buffer, the values are associated with `bufferVar.statuses` for the readings.

Example

This example enables limits 1 and 2 for DC volt, measurements. Limit 1 is checking for readings to be between 3 and 5 volts while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail will be 1 afterwards. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
dmm.func = "dcvolts"                 -- set the DMM for DC volts functionality
dmm.reset("active")                  -- reset DC volts to default settings
dmm.range = 10                       -- set the range to 10 volts
dmm.nplc = 0.1                       -- set the nplc to 0.1
dmm.limit[1].autoclear = dmm.OFF     -- disable auto clearing for limit 1
dmm.limit[1].high.value = 5          -- set high limit on 1 to fail if reading
                                     -- exceeds 5 volts
dmm.limit[1].low.value = 3           -- set low limit on 1 to fail if reading
                                     -- is less than 3 volts
dmm.limit[1].enable = dmm.ON         -- enable limit 1 checking for DC volt
                                     -- measurements
dmm.limit[2].autoclear = dmm.OFF     -- disable auto clearing for limit 2
dmm.limit[2].high.value = 7          -- set high limit on 2 to fail if reading
                                     -- exceeds 7 volts
dmm.limit[2].low.value = 1           -- set low limit on 2 to fail if reading
                                     -- is less than 1 volts
dmm.limit[2].enable = dmm.ON         -- enable limit 2 checking for DC volt
                                     -- measurements
dmm.measurecount = 50                -- set the measure count to 50
LimitBuffer = dmm.makebuffer(100)    -- create a reading buffer that can store
                                     -- 100 readings
dmm.measure(LimitBuffer)             -- take 50 readings and store them in
                                     -- LimitBuffer
                                     -- then check if any of the 50 readings
                                     -- were outside of the limits
print("limit 1 high fail = " .. dmm.limit[1].high.fail)
print("limit 1 low fail = " .. dmm.limit[1].low.fail)
print("limit 2 high fail = " .. dmm.limit[2].high.fail)
print("limit 2 low fail = " .. dmm.limit[2].low.fail)
dmm.limit[1].clear()                 -- clear limit 1 conditions
dmm.limit[2].clear()                 -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 high fail = 0
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 high fail = 1
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit limit 1 and 2 low values (a 0.5 volts reading would cause this condition or a reading less than 1):

```
limit 1 high fail = 0
limit 1 low fail = 1
limit 2 high fail = 0
limit 2 low fail = 1
```

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].autoclear (on page 8-191)

dmm.limit[Y].low.value

The low limit value for limit *Y* when `dmm.limit[Y].enable` is set to `dmm.ON`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	limit 1: -1.000000E+00 limit 2: -2.000000E+00

Usage

```
value = dmm.limit[Y].low.value
dmm.limit[Y].low.value = value
```

value	The low limit value of limit *Y*; The valid range is -4294967295 to +4294967295
Y	Limit number 1 or 2

Details

This attribute is valid for all functions except `"continuity"` and `"nofunction"`. A nil response and an error is generated if the command is received when `dmm.func` is set to either of these functions.

This attribute specifies or queries the low limit value of limit *Y*. When limit *Y* testing is enabled (`dmm.limit[Y].enable` = 1), a fail indication occurs when the measurement value is less than this value.

You may set or get the value regardless if the limit is set to a digital I/O trigger stimulus:

- `dmm.trigger.EVENT_LIMIT1_HIGH`
- `dmm.trigger.EVENT_LIMIT1_LOW`
- `dmm.trigger.EVENT_LIMIT2_HIGH`
- `dmm.trigger.EVENT_LIMIT2_LOW`

Example

This example enables limits 1 and 2 for DC volt, measurements. Limit 1 is checking for readings to be between 3 and 5 volts while limit 2 is checking for the readings to be between 1 and 7 volts. The auto clear feature is disabled, so if any reading is outside these limits, the corresponding fail will be 1 afterwards. Therefore, if any one of the fails is 1, analyze the reading buffer data to find out which reading failed the limits.

```
dmm.func = "dcvolts"                -- set the DMM for DC volts functionality
dmm.reset("active")                 -- reset DC volts to default settings
dmm.range = 10                      -- set the range to 10 volts
dmm.nplc = 0.1                      -- set the nplc to 0.1
dmm.limit[1].autoclear = dmm.OFF    -- disable auto clearing for limit 1
dmm.limit[1].high.value = 5         -- set high limit on 1 to fail if reading
                                       -- exceeds 5 volts
dmm.limit[1].low.value = 3          -- set low limit on 1 to fail if reading
                                       -- is less than 3 volts
dmm.limit[1].enable = dmm.ON        -- enable limit 1 checking for DC volt
                                       -- measurements
dmm.limit[2].autoclear = dmm.OFF    -- disable auto clearing for limit 2
dmm.limit[2].high.value = 7         -- set high limit on 2 to fail if reading
                                       -- exceeds 7 volts
dmm.limit[2].low.value = 1          -- set low limit on 2 to fail if reading
                                       -- is less than 1 volts
dmm.limit[2].enable = dmm.ON        -- enable limit 2 checking for DC volt
                                       -- measurements
dmm.measurecount = 50               -- set the measure count to 50
LimitBuffer = dmm.makebuffer(100)   -- create a reading buffer that can store
                                       -- 100 readings
dmm.measure(LimitBuffer)            -- take 50 readings and store them in
                                       -- LimitBuffer then check if any
                                    -- of the 50 readings were
                                       -- outside of the limits
print("limit 1 high fail = " .. dmm.limit[1].high.fail)
print("limit 1 low fail = " .. dmm.limit[1].low.fail)
print("limit 2 high fail = " .. dmm.limit[2].high.fail)
print("limit 2 low fail = " .. dmm.limit[2].low.fail)
dmm.limit[1].clear()                -- clear limit 1 conditions
dmm.limit[2].clear()                -- clear limit 2 conditions
```

Sample output that shows all readings are within limit values (all readings between 3 and 5 volts):

```
limit 1 high fail = 0
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit 1 high values (a 6 volt reading would cause this condition or a reading greater than 5 but less than 7.):

```
limit 1 high fail = 1
limit 1 low fail = 0
limit 2 high fail = 0
limit 2 low fail = 0
```

Sample output showing at least one reading failed limit limit 1 and 2 low values (a 0.5 volts reading would cause this condition or a reading less than 1):

```
limit 1 high fail = 0
limit 1 low fail = 1
limit 2 high fail = 0
limit 2 low fail = 1
```

Also see

Reading buffers (on page 3-55, on page 3-49)
digio.trigger[N].stimulus (on page 8-127)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.limit[Y].enable (on page 8-193)
dmm.limit[Y].high.value (on page 8-197)
dmm.limit[Y].low.fail (on page 8-199)

dmm.linesync

Selects if line sync is used during the measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
state = dmm.linesync
dmm.linesync = state
```

state	Enable line sync (`dmm.ON` or `1`). Disable line sync (`dmm.OFF` or `0`).

Details

This attribute is only valid when `dmm.func` is set to `"commonsideohms"`, `"continuity"`, `"dccurrent"`, `"dcvolts"`, `"fourwireohms"`, `"temperature"`, or `"twowireohms"`. All other functions generate an error when written and return `nil` when queried.

When `dmm.linesync` is enabled, measurements are initiated at the first positive-going zero crossing of the power line cycle after the trigger.

The line sync setting is saved with the `dmm.func` function setting, so if you use another function, then return to a previous function, the line sync you set previously is retained.

Example

`dmm.func = "fourwireohms"` `dmm.linesync = dmm.ON`	Enables line sync for the "fourwireohms" function.

Also see

Line cycle synchronization (on page 4-4)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.makebuffer()

Creates a user buffer for storing readings. Reading buffers are allocated dynamically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Reset Recall setup	Create configuration script Save setup	None

Usage

`bufferVar = dmm.makebuffer(bufferSize)`

bufferVar	The variable name for the buffer being created
bufferSize	Maximum number of readings that the buffer can store

Details

To be able to store readings, a reading buffer needs to be created. Once created, the reading buffer can be used to store readings from `dmm.measure()` command and from scanning (`scan.execute()` or `scan.background()`)

To delete a buffer, set *bufferVar* to `nil`.

Once a buffer is created, the attributes that can be accessed are:

- *bufferVar*.appendmode = `1` (ON) or `0` (OFF); default is zero (0) over a bus interface, and 1 for buffers created on the front panel.
- *bufferVar*.`basetimeseconds` returns the seconds for reading buffer entry 1 (read-only attribute).
- *bufferVar*.`basetimefractional` returns the seconds and fractional seconds for reading buffer entry 1 (read-only attribute).
- *bufferVar.*`capacity` returns the overall buffer size.
- *bufferVar*.`collecttimestamps` = `1` (ON) or `0` (OFF); default is `1`.
- *bufferVar*.`collectchannels` = `1` (ON) or `0` (OFF); default is `1`.
- *bufferVar*.n returns the number of readings currently stored in the buffer.
- *bufferVar*.`timestampresolution` returns the resolution of the time stamping (read-only attribute).

The following buffer bits indicate buffer statuses:

- `dmm.buffer.LIMIT1_LOW_BIT or 1`
- `dmm.buffer.LIMIT1_HIGH_BIT or 2`
- `dmm.buffer.LIMIT2_LOW_BIT` or `4`
- `dmm.buffer.LIMIT2_HIGH_BIT` or `8`
- `dmm.buffer.MEAS_OVERFLOW_BIT` or `64`
- `dmm.buffer.MEAS_CONNECT_QUESTION_BIT` or `128`

To see readings in buffer:

`printbuffer(x, y, bufferVar)`

Where *x* and *y* represent the reading numbers desired.

To see readings, channels, and units:

`printbuffer(x, y, bufferVar, bufferVar.channels, bufferVar.units)`

Where *x* and *y* represent reading numbers desired.

To see time stamps in buffer:

```
bufferVar.collecttimestamps = 1
print(x, y, bufferVar, bufferVar.timestamps)
```

Where x and y represent readings and time stamps for elements x to y.

To see seconds, fractional seconds, and relative time stamps:

```
bufferVar.collecttimestamps = 1
printbuffer(x,y, bufferVar.seconds)
printbuffer(x,y, bufferVar.fractionalseconds)
printbuffer(x,y, bufferVar.relativetimestamps)
```

CAUTION

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

Example 1

`bufferVar = dmm.makebuffer(300)`	Creates a user reading buffer named `bufferVar` with a capacity of 300.

Example 2

`dmm.measurecount = 10` `dmm.measure(bufferVar2)` `printbuffer(1, bufferVar2.n, bufferVar2)` `bufferVar2 = nil`	Take ten measurements on the active function and store them in the reading buffer, bufferVar2. View those ten readings. Delete bufferVar2. Sample output (actual output depends on how the active function is configured and what you are measuring): `1.134154698e+01, 1.132708486e+01, 1.134213865e+01, 1.134037749e+01, 1.132735758e+01, 1.134099844e+01, 1.133705087e+01, 1.132571507e+01, 1.134000616e+01, 1.133721111e+01`

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.measure() (on page 8-213)
printbuffer() (on page 8-307)
scan.background() (on page 8-323)
scan.execute() (on page 8-327)

dmm.math.enable

Enable or disable math operation on measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
value = dmm.math.enable
dmm.math.enable = value
```

value	The math enable setting: • Enable: dmm.ON or 1 • Disable: dmm.OFF or 0

Details

This attribute is not available for `"nofunction"`. If you write this attribute for `"nofunction"`, an error message is generated.

When this attribute is set to `dmm.ON`, the math operation specified by math format attribute (`dmm.math.format`) will be performed before completing a measurement.

Example

`dmm.func = "dcvolts"` `dmm.reset("active")` `dmm.measurecount = 5` `MathBuffer = dmm.makebuffer(100)`	Configure the DMM for DC volts. Reset DC volts to the default settings. Set the measure count to 5. Create a reading buffer named MathBuffer that can store 100 readings.
`MathBuffer.appendmode = 1` `dmm.measure(MathBuffer)` `dmm.math.format = dmm.MATH_MXB` `dmm.math.mxb.mfactor = 1e6` `dmm.math.mxb.bfactor = 0` `dmm.math.mxb.units = "["` `dmm.math.enable = dmm.ON`	Set the buffer to append readings. Take 5 readings and store them in MathBuffer with no math operation. Enable math operations for mx+b operation, with m set to 1e6 and b set to 0, with units set to micro .
`dmm.measure(MathBuffer)`	Store the 5 additional readings in MathBuffer with math operations enabled.
`printbuffer(1, 5, MathBuffer)` `printbuffer(6, MathBuffer.n, MathBuffer)`	View the readings with and without math operation.
`dmm.measurecount = 1` `for x = 1, 3 do` `    print(dmm.measure())` `end`	Take 3 additional math readings without using the buffer.
	Sample output assuming no load was connected to DMM: Readings with no math operation: `3.898423119e-07, 4.066727213e-07,` `5.122452892e-07,` `4.724643216e-07,` `4.770544332e-07` Readings with math operation: `5.061251403e-01, 4.158529446e-01,` `5.504962196e-01,` `3.821921259e-01,` `6.132277455e-01` `5.367258847e-01` `6.040475222e-01` `6.132277455e-01`

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.format (on page 8-208)

dmm.math.format

Specifies the math operation to perform on measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	2 (dmm.MATH_PERCENT)

Usage

```
state = dmm.math.format
dmm.math.format = state
```

state	Math operation to be performed on measurements: • dmm.MATH_NONE or 0 • dmm.MATH_MXB or 1 • dmm.MATH_PERCENT or 2 • dmm.MATH_RECIPROCAL or 3

Details

This is not available for `"nofunction"`. If this command is queried when `"nofunction"` is selected, nil is returned. If it is written when `"nofunction"` is selected, an error is returned.

If you set this attribute to `dmm.MATH_NONE`, math operation is disabled, even if math operation (`dmm.math.enable`) is enabled.

Use a setting of `dmm.MATH_MXB` to have

Equation 1: S3700A dmm_math_mxb_formula

$$Y = mX + b$$

where:

- **X** is the normal measurement
- **m** is user-entered constant for scale factor (`dmm.math.mxb.mfactor`)
- **b** is user-entered constant for offset (`dmm.math.mxb.bfactor`)
- **Y** is the result

If you are using relative offset measurement control (`dmm.rel.enable`), the relative offset reading is used for X.

Use a setting of `dmm.MATH_PERCENT` to have:

Equation 2: dmm.MATH_PERCENT Equation

$$Percent = \frac{Input - Reference}{Reference} \times 100\%$$

where:

- **Input** is the normal measurement (if using `dmm.rel.enable`, it will be the relative offset value)
- **Reference** is user entered constant (`dmm.math.percent`)
- **Percent** is the result

Use a setting of `dmm.MATH_RECIPROCAL` for 1/X operation, where `X` is normal or the measurement value with relative offset applied.

The desired math operation is performed before any of the enabled limit testing.

Example

`dmm.math.format = dmm.MATH_RECIPROCAL` `dmm.math.enable = dmm.ON`	Enables the reciprocal operation on measurements

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.enable (on page 8-206)
dmm.math.percent (on page 8-212)
dmm.math.mxb.bfactor (on page 8-209)
dmm.math.mxb.mfactor (on page 8-210)
dmm.rel.enable (on page 8-225)

dmm.math.mxb.bfactor

Specifies the offset for the y = mx + b operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0.000000e+00

Usage

```
value = dmm.math.mxb.bfactor
dmm.math.mxb.bfactor = value
```

`value`	The offset for the y = mx + b operation; the valid range is -4294967295 to +4294967295

Details

This is not available for `"nofunction."`. If command is queried when `"nofunction"` is selected, nil is returned. If it is written when `"nofunction"` is selected, an error is returned.

This attribute specifies the offset (b) for an mx + b operation.

Example

`dmm.math.mxb.bfactor = 50`	Sets the offset for mx +b operation to 50.

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.format (on page 8-208)
dmm.math.mxb.mfactor (on page 8-210)

dmm.math.mxb.mfactor

Specifies the scale factor for the y = mx + b operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+00

Usage

```
value = dmm.math.mxb.mfactor
dmm.math.mxb.mfactor = value
```

value	The scale factor; valid range is -4294967295 to +4294967295

Details

This is not available for `"nofunction"`. If command is queried when `"nofunction"` is selected, nil is returned. If it is written when `"nofunction"` is selected, an error is returned.

This attribute represents the scale factor (m) for an mx + b operation.

Example

`dmm.math.mxb.mfactor = 0.80`	Sets the scale factor for the mx +b operation to 0.80.

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.format (on page 8-208)
dmm.math.mxb.bfactor (on page 8-209)

dmm.math.mxb.units

Specifies the unit character for the y = mX + b operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	X

Usage

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

value	The unit character for the y = mx + b operation. Valid values are: • A to Z • [(left bracket) for the micro (µ) symbol •] (right bracke) for the ohm (Ω) symbol • \\ (two backslashes) for the degree (°) symbol

Details

This attribute is not available for the `"nofunction"` selection. If the command is queried when `"nofunction"` is selected, `nil` is returned. If it is written when `"nofunction"` is selected, an error is returned.

This attribute represents the unit character to use when the math format is set for mx + b (`dmm.math.format = dmm.MATH_MXB`).

Example

`dmm.math.mxb.units = "Q"`	Sets the units for the mX +b operation to `"Q"`.

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.format (on page 8-208)

dmm.math.percent

Specifies the constant to use for the percent operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+00

Usage

```
value = dmm.math.percent
dmm.math.percent = value
```

value	The constant for the percent operation; the range is -4294967295 to +4294967295

Details

This is not available for `"nofunction"`. If command is queried when `"nofunction"` is selected, `nil` is returned. If it is written when `"nofunction"` is selected, an error is returned.

This attribute represents the constant to use for percent when `dmm.math.format = dmm.MATH_PERCENT`.

Example 1

`dmm.math.percent = 1250`	Set constant for percent operation to 1250.
`dmm.math.percent = dmm.measure()`	Acquire the percent constant.

Example 2

```
dmm.func = "dcvolts"
dmm.reset("active")
dmm.math.format = dmm.MATH_PERCENT
dmm.measurecount = 1
dmm.math.percent = dmm.measure()
dmm.math.enable = dmm.ON
dmm.measurecount = 5
MathBuffer = dmm.makebuffer(100)
dmm.measure(MathBuffer)
printbuffer(1, MathBuffer.n,
   MathBuffer)
dmm.measurecount = 1
for x = 1, 3 do
    print(dmm.measure())
end
```

Configure the DMM for DC volts and reset the DC volts 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 DMM:

```
2.717115286e+01, 1.259150986e+01,
    1.259150986e+01, 9.277954635e+00,
    3.313555227e+01

1.292338066e+01
2.452080209e+01
1.557421984e+01
```

Also see

Math calculations (on page 4-43)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.math.format (on page 8-208)

dmm.measure()

Returns the last reading of the measurement process without using the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = dmm.measure()
reading = dmm.measure(bufferVar)
```

reading	The last reading of the measurement process
bufferVar	A previously created reading buffer where all readings are stored

Details

This is not available for `"nofunction"`. If the command is queried when `"nofunction"` is selected, `nil` is returned. If it is written when `"nofunction"` is selected, an error is returned.

When a reading buffer is used with a command or action that involves taking multiple readings, such as `dmm.measure` or scanning, all readings are available in the reading buffer. However, only the last reading is returned as a reading with the command.

You can also use a reading buffer to store additional information that is acquired while making a measurement.

The `dmm.measurecount` attribute determines how many measurements are performed. When you use a buffer, it also determines if the reading buffer has enough room to store the requested readings. The amount of room is based on readings already stored in the buffer (*`bufferVar`*`.n`), the capacity of the buffer (*`bufferVar`*`.capacity`), and the append mode of the reading buffer (*`bufferVar`*`.appendmode`). If the append mode is set to 0, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to 1, any stored readings remain in the buffer and new ones are added to the buffer after the stored ones.

Example

```
DCVBuffer = dmm.makebuffer(100)
dmm.func = "dcvolts"
dmm.measurecount = 100
dmm.measure(DCVBuffer)
```

Performs 100 DC voltage measurements and stores them in a buffer called `DCVBuffer`.

Also see

Reading buffers (on page 3-55, on page 3-49)
bufferVar.appendmode (on page 8-18)
bufferVar.capacity (on page 8-21)
bufferVar.n (on page 8-29)
dmm.makebuffer() (on page 8-204)
dmm.measurecount (on page 8-214)
dmm.measurewithtime() (on page 8-215)

dmm.measurecount

The number of measurements to take when a measurement is requested by a DMM measure command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1

Usage

```
count = dmm.measurecount
dmm.measurecount = count
```

count	The number of measurements to take when a DMM measure function is used (maximum 450,000)

Details

This attribute controls the number of measurements taken any time a measurement is requested (through `dmm.measure`, `dmm.measurewithtime`, or the front panel MEASURE menu option). When using a reading buffer with a measure command, the count also controls the number of readings to be stored.

It has no effect on the trigger model, and the trigger model does not affect this setting.

This setting is applied to all functions (the setting is not related to a specific function).

Example

`DMMbuffer = dmm.makebuffer(500)` `dmm.measure(bufferVar)` `dmm.measurecount = 50`	Create a reading buffer called DMMbuffer that can store 500 readings. Store 50 readings in DMMbuffer. Set the measure count of the DMM to 50.

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.autodelay (on page 8-154)
dmm.makebuffer() (on page 8-204)
dmm.measure() (on page 8-213)
dmm.measurewithtime() (on page 8-215)

dmm.measurewithtime()

Returns the last actual measurement and time information in UTC format without using the trigger model. You can also use a reading buffer to store additional information that is acquired while making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = dmm.measurewithtime()
reading, seconds, fractional = dmm.measurewithtime(bufferVar)
```

reading	The last reading of the measurement process
seconds	Seconds in UTC format
fractional	Fractional seconds
bufferVar	A previously created reading buffer variable in which all readings are stored

Details

This is not available for `"nofunction"`. If the command is queried when `"nofunction"` is selected, `nil` is returned. If it is written when `"nofunction"` is selected, an error is returned.

When a reading buffer is used with a command or action that involves taking multiple readings, such as `dmm.measure` or scanning, all readings are available in the reading buffer. However, only the last reading and time information (seconds and fractional seconds) is returned as a reading with the command.

You can also use a reading buffer to store additional information that is acquired while making a measurement.

The `dmm.measurecount` attribute determines how many measurements are performed. When you use a buffer, it also determines if the reading buffer has enough room to store the requested readings. The amount of room is based on readings already stored in the buffer (*`bufferVar`*`.n`), the capacity of the buffer (*`bufferVar`*`.capacity`), and the append mode of the reading buffer (*`bufferVar`*`.appendmode`). If the append mode is set to 0, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to 1, any stored readings remain in the buffer and new ones are added to the buffer after the stored ones.

Example

```
DCVbuffer = dmm.makebuffer(100)
dmm.func = "dcvolts"
dmm.measurecount = 100
reading, seconds, fractional = dmm.measurewithtime(DCVbuffer)
print(reading, seconds, fractional)
```

Create a reading buffer.
Perform 100 DC voltage measurements.
Store the measurements in a buffer called `DCVbuffer`.
Print the last measurement and time information in UTC format, which will look similar to:

```
-1.064005867e-02   1.779155900e+07   1.245658350e-01
```

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.makebuffer() (on page 8-204)
dmm.measure() (on page 8-213)
dmm.measurecount (on page 8-214)

dmm.measurewithptp()

This function returns the last actual measurement and time information in PTP format without using the trigger model. You can also use a reading buffer to store additional information that is acquired while making a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading, seconds, fractional = dmm.measurewithptp()
reading, seconds, fractional = dmm.measurewithptp(bufferVar)
```

reading	The last reading of the measurement process
seconds	Seconds in PTP format
fractional	Fractional seconds
bufferVar	A previously created reading buffer variable in which all readings are stored

Details

This is not available for `"nofunction"`. If the command is queried when `"nofunction"` is selected, `nil` is returned. If it is written when `"nofunction"` is selected, an error is returned.

When a reading buffer is used with a command or action that involves taking multiple readings, such as `dmm.measure` or scanning, all readings are available in the reading buffer. However, only the last reading and time information (seconds and fractional seconds) is returned as a reading with the command.

You can also use a reading buffer to store additional information that is acquired while making a measurement.

The `dmm.measurecount` attribute determines how many measurements are performed. When you use a buffer, it also determines if the reading buffer has enough room to store the requested readings. The amount of room is based on readings already stored in the buffer (*bufferVar*.n), the capacity of the buffer (*bufferVar*`.capacity`), and the append mode of the reading buffer (*bufferVar*`.appendmode`). If the append mode is set to 0, any stored readings in the buffer are cleared before new ones are stored. If append mode is set to 1, any stored readings remain in the buffer and new ones are added to the buffer after the stored ones.

Example

```
DCVbuffer = dmm.makebuffer(100)
dmm.func = "dcvolts"
dmm.measurecount = 100
reading, seconds, fractional = dmm.measurewithptp(DCVbuffer)
print(reading, seconds, fractional)
```

Create a reading buffer.
Perform 100 DC voltage measurements.
Store the measurements in a buffer called `DCVbuffer`.
Print the last measurement and time information in PTP format, which will look similar to:

```
-1.064005867e-02   1.779155900e+07   1.245658350e-01
```

Also see

Reading buffers (on page 3-55, on page 3-49)
dmm.makebuffer() (on page 8-204)
dmm.measure() (on page 8-213)
dmm.measurecount (on page 8-214)

dmm.nplc

The integration rate in line cycles for the DMM for the function selected by `dmm.func`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+000

Usage

```
value = dmm.nplc
dmm.nplc = value
```

value	The integration rate in line cycles: • 60 Hertz: 0.0005 to 15 • 50 Hertz: 0.0005 to 12

Details

This attribute is not applicable for `"frequency"`, `"period"`, and `"nofunction"`. If you query this attribute for one of these functions, `nil` is returned. Note that "continuity" is fixed at 6.000000E-003 and cannot be changed.

The setting for NPLC may be adjusted based on what the DMM supports. Therefore, after setting the NPLC, query the value to see if it was adjusted.

NOTE

For `dmm.nplc` settings that are less than 0.2, sending `dmm.AUTOZERO_ONCE` results in significant delays. For example, the delay time at an NPLC of 0.0005 is 2.75 s. The delay time at 0.199 is 5.45 s.

An error is generated if the command is used when `dmm.func` is set to `"frequency"`, `"period"`, `"continuity"`, or `"nofunction"`.

The NPLC setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the NPLC setting you set previously is retained.

Example

`dmm.func = "twowireohms"` `dmm.nplc = 0.5` `dmm.func = "dcvolts"` `dmm.nplc = 0.1`	Set the NPLC for 2-wire ohms to 0.5, then set the NPLC for DC volts to 0.1.

Also see

dmm.aperture (on page 8-150)
dmm.autozero (on page 8-157)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.offsetcompensation

The offset compensation setting for the DMM for the function selected by `dmm.func`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF) for "commonsideohms" and "fourwireohms" 1 (dmm.ON) for "temperature"

Usage

```
state = dmm.offsetcompensation
dmm.offsetcompensation = state
```

state	The offset compensation setting: • Enable: dmm.ON or 1 • Disable: dmm.OFF or 0

Details

The command applies when `dmm.func` is set to `"fourwireohms"`, `"commonsideohms"` or `"temperature"`.

When `dmm.func = "temperature"`, this attribute applies only when the transducer type is 3- or 4-wire RTD. Set this command as you would for 4-wire ohm measurements.

This command is automatically set to `dmm.ON` when `dmm.drycircuit` is set to `dmm.ON` and `dmm.func = "fourwireohms"` or `"commonsideohms"`.

The offset compensation setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"fourwireohms"`, `"commonsideohms"` or `"temperature"`, the offset compensation setting you set previously is retained.

If you query this attribute and the function is not `"fourwireohms"`, `"commonsideohms"`, or `"temperature"`, nil is returned.

Example 1

`dmm.func = "fourwireohms"` `dmm.offsetcompensation = dmm.ON`	Enable offset compensation for 4-wire ohms.

Example 2

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.offsetcompensation = dmm.OFF`	Disable offset compensation for 3-wire RTD temperature measurements.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.open()

Opens the specified channel or channel pattern.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`dmm.open(channelList)`

channelList	A string that lists the channel or channel pattern to open

Details

This command allows you to separate the opening and closing of channels and analog backplane relays when measuring. You can execute any number of commands between the open and close commands to meet your application needs.

The configuration (`dmm.getconfig()`) associated with the specified channel dictates whether a paired channel is open or not. For channel patterns, the channels associated with it are opened. A channel pattern includes a paired channel for multi-wire measurement if a channel is configured that way when the pattern is created (see commands `channel.setpole()` and `channel.pattern.setimage()`)

The configuration (`dmm.getconfig()`) dictates whether analog backplane relays 1 and 2 are opened and if a paired channel is opened. The `dmm.open()` function does not use the analog backplane relays specified by the `channel.setbackplane()` function or pole settings set by the `channel.setpole()` function.

An error is generated and the channels do not open if:

- An empty parameter string is specified.
- The specified channel or channel pattern is invalid.
- A channel number does not exist for installed card in slot specified.
- A slot is empty.
- The channel pattern does not exist.
- The channel does not support being closed (for example, a digital I/O channel).
- The channel is paired with another bank for a multi-wire application.
- The channel configuration is "nofunction".
- More than one channel or channel pattern is specified in the parameter.

Example 1

```
reset()
channel.setpole("slot2" , 4)
channel.pattern.setimage("2005, 2911, 2922", "Chan5_4W")
dmm.setconfig("Chan5_4W", "fourwireohms")
dmm.open("Chan5_4W")
print(channel.pattern.getimage("Chan5_4W"))
```

Assume a Model 3721 is installed in slot 2. Reset the instrument. Configure the slot 2 channels for 4-pole operation. Create a pattern called Chan5_4W. Assign 4-wire ohms configuration to the Chan5_4W pattern. Open the channels associated with Chan5_4W and display image of the Chan5_4W. Output: `2005(2025),2911,2922`

Example 2

```
dmm.setconfig("slot3", "dcvolts")
dmm.close("3030")
print(channel.getclose("slot3"))
dmm.open("3030")
print(channel.getclose("slot3"))
dmm.close("3031")
print(channel.getclose("slot3"))
dmm.open("3031")
print(channel.getclose("slot3"))
```

Assume a 3720 installed in slot 3.
Set the configuration for DC volts.
Close and open the channels.
Output:

```
3030;3911
nil
3031;3921
nil
```

Also see

channel.getclose() (on page 8-61)
channel.pattern.getimage() (on page 8-81)
channel.pattern.setimage() (on page 8-82)
channel.setbackplane() (on page 8-90)
channel.setpole() (on page 8-101)
dmm.close() (on page 8-167)
dmm.getconfig() (on page 8-189)

dmm.opendetector

Determines if the detection of open leads is enabled or disabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF) for "commonsideohms" 1 (dmm.ON) for "fourwireohms" and "temperature"

Usage

```
state = dmm.opendetector
dmm.opendetector = state
```

state	Enable open lead detector (`dmm.ON` or `1`) Disable open lead detector (`dmm.OFF` or `0`)

Details

The command applies when `dmm.func` is set to `"fourwireohms"`, `"commonsideohms"`, or `"temperature"`.

When `dmm.func` is set to temperature, the open detector setting is only used when the transducer type is thermocouple. For all other transducer types, it is set, but not used until the transducer type is set to thermocouple.

The open detector setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"fourwireohms"`, `"commonsideohms"`, or `"temperature"`, the open detector setting you set previously is retained.

An error is generated if `dmm.func` is set to any function other than `"fourwireohms"`, `"commonsideohms"`, or `"temperature"`. If you query the setting for any other function, `nil` is returned.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMOCOUPLE` `dmm.opendetector = dmm.ON`	Enable the thermocouple open detector.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.range

Indicates the range of DMM for the selected function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	See **Details**

Usage

```
value = dmm.range
dmm.range = value
```

value	The range for the function selected by `dmm.func`

Details

Set this value to the expected measurement value and the instrument will select the range appropriate to measure that value. Setting the range with this attribute will automatically disable the autorange setting (`dmm.autorange` command).

The instrument selects the range to best match the expected measure value for the functions, as shown below.

Ranges and defaults		
If dmm.func is…	**The range is…**	**The default is…**
`"dcvolts"`	0 to 303	303
`"acvolts"`	0 to 303	10
`"dccurrent"`	0 to 3.1	3.1
`"accurrent"`	0 to 3.1	3.1
`"twowireohms"`	0 to 120e6	1000
`"fourwireohms"`	0 to 120e6	1000
`"commonsideohms"`	0 to 120e6	1000

The range setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the range settings you set previously are retained.

If you query the range when the selected function does not have a range associated with it, `nil` is returned.

An error is generated if:

- The `dmm.range` is received when `dmm.func` is `"temperature"`, `"frequency"`, `"period"`, `"continuity"`, or `"nofunction"`.
- If *value* does not make sense for selected function.

Example

`dmm.func = "dcvolts"` `dmm.range = 5` `dmm.func = "twowireohms"` `dmm.range = 35000` `print(dmm.range)`	Set the range for DC volts to 10. Select a range on 2-wire ohms suitable for measuring 35000. View the selected range. Output: `1.000000000e+05`

Also see

dmm.autorange (on page 8-155)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.reset() (on page 8-228)

dmm.refjunction

The type of the thermocouple reference junction.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.REF_JUNCTION_INTERNAL) for "temperature"

Usage

```
state = dmm.refjunction
dmm.refjunction = state
```

state	The reference junction type: • dmm.REF_JUNCTION_SIMULATED or 0 • dmm.REF_JUNCTION_INTERNAL or 1 • dmm.REF_JUNCTION_EXTERNAL or 2

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return `nil` when queried.

This attribute only applies when the transducer type is set to thermocouple. For all other transducer types, the reference junction may be set, but it is not used until the transducer type is set to thermocouple.

The reference junction setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the reference junction settings you set previously are retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMOCOUPLE` `dmm.refjunction = dmm.REF_JUNCTION_SIMULATED`	Enables the simulated thermocouple reference junction.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)

dmm.rel.acquire()

Acquires an internal measurement to store as the relative level value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`relativeValue = dmm.rel.acquire()`

relativeValue	The internal measurement acquired for the relative offset level value

Details

This attribute is not applicable for `"continuity"` and `"nofunction"`.

This function triggers the DMM to take a new measurement for the selected function. This measurement is then stored as the new relative offset level setting.

After executing this command, use the `dmm.rel.level` attribute to see the last relative level value that was acquired or set by the user. Setting the relative level value with the acquire function does not use the math, limit, and filter settings. It is a calibrated reading as if these settings are disabled.

If error occurs during the reading, nil is returned.

An error is generated if:

- `dmm.func` is set to `"continuity"` or `"nofunction"`.
- The DMM is unable to take the measurement.

When an error occurs, the relative offset level setting maintains the last valid setting.

Example

`dmm.func = "temperature"` `rel_value = dmm.rel.acquire()`	Acquires a relative offset level value for temperature.

Also see

dmm.func (on page 8-187)
dmm.rel.level (on page 8-226)

dmm.rel.enable

Enables or disables relative measurement control for the function selected by `dmm.func`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.OFF)

Usage

```
value = dmm.rel.enable
dmm.rel.enable = value
```

value	The setting: • Enable: dmm.ON or 1 • Disable: dmm.OFF or 0

Details

This attribute is not available if `dmm.func` is set to `"continuity"` or `"nofunction"`. If you query this attribute when either of these functions are selected, nil is returned.

When relative measurements are enabled, all subsequent measured readings are offset by the relative offset specified by `dmm.rel.level`. Each returned measured relative reading will be the result of the following calculation:

Relative reading = Actual measured reading – Relative offset value

If you change functions with `dmm.func`, the relative enable setting changes to the enable setting for that function.

The relative enable setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the relative enable settings you set previously are retained.

An error is generated if:

- `dmm.func` is set to `"continuity"` or `"nofunction"`.
- If the value is out of range for the selected function.

Example

`dmm.func = "accurrent"` `dmm.rel.acquire()` `dmm.rel.enable = dmm.ON`	Enables the relative measurements for AC current and uses the acquire command to set the relative level attribute.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.rel.acquire() (on page 8-224)
dmm.rel.level (on page 8-226)

dmm.rel.level

The offset value for relative measurements for the function selected by `dmm.func`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0.000000E+000

Usage

```
value = dmm.rel.level
dmm.rel.level = value
```

value	The relative offset level setting

Details

This attribute is not available if `dmm.func` is set to `"continuity"` or `"nofunction"`. If you query this attribute when either of these functions are selected, nil is returned.

When relative measurements are enabled (as set by `dmm.rel.enable`), all subsequent measured readings are offset by the specified relative offset value. Specifically, each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading – Relative offset value

Changing functions with `dmm.func` reflects the relative level offset setting for that function.

The relative offset level setting is saved with the `dmm.func` function setting, so if you use another function, then return to the previous function, the relative offset level settings you set previously are retained.

NOTE

To set the relative offset level to include math, limits, and filter operations (if enabled) set `dmm.rel.level` to `dmm.measure()`. However, these operations are not used if you use the `dmm.rel.acquire()` function to set the relative offset level, even if the operations are enabled.

An error is generated:

- If `dmm.func` is set to `"continuity"` or `"nofunction"`.
- If the value is out of range for the selected function.

Example

`dmm.func = "accurrent"` `dmm.rel.level = dmm.measure()` `rel_value = dmm.measure()` `dmm.rel.level = rel_value` `dmm.func = "temperature"` `rel_value = dmm.rel.acquire()`	Perform an AC current measurement and use it as the relative offset value. Take a measurement and store it in the variable `rel_value`. Use the `rel_value` to set the relative level attribute. Acquire a relative offset level value for temperature.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.measure() (on page 8-213)
dmm.rel.acquire() (on page 8-224)
dmm.rel.enable (on page 8-225)

dmm.reset()

Resets the DMM functions and attributes in the instrument, as indicated by the parameter.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.reset(scope)
```

scope	A string equaling "active" to set the active function only to factory default settings or "all" to set all functions back to factory default settings

Details

When the scope is set to active, this command resets the DMM attributes for the active function to factory default values. The settings for other functions are unchanged.

When the scope is set to all, this command resets the DMM functions and attributes to factory default settings.

This function does not affect the DMM configurations (`dmm.setconfig()` and `dmm.getconfig()`).

The factory default settings are:

- The selected DMM function is set to `"dcvolts"`.
- The DMM settings are set to the defaults for `"dcvolts"`.
- All attribute settings for other functions are set to factory default settings.

NOTE

To reset the entire instrument to factory default settings, use the `reset` command.

Example

`dmm.func = "temperature"` `dmm.reset("active")` `print(dmm.func)` `dmm.reset("all")` `print(dmm.func)`	Set the DMM function to temperature. Perform a reset on temperature only. Check the function after resetting only temperature. Perform a reset on all DMM functions. Check the function after resetting all DMM functions. Output: `temperature` `dcvolts`

Also see

dmm.func (on page 8-187)
dmm.getconfig() (on page 8-189)
dmm.setconfig() (on page 8-237)
reset() (on page 8-317)

dmm.rtdalpha

The user type RTD alpha value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	3.850550E-03

Usage

```
value = dmm.rtdalpha
dmm.rtdalpha = value
```

value	The RTD alpha value; the range is 0 to 0.01

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return `nil` when queried.

This setting only applies when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting is set but not used until the transducer type is set to an RTD type.

Changing functions with `dmm.func` reflects the setting for that function.

The RTD alpha setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the RTD alpha setting you set previously is retained.

NOTE

The following attributes share common settings and apply to both 3 and 4-wire RTDs: `dmm.rtdalpha`, `dmm.rtdbeta`, `dmm.rtddelta`, and `dmm.rtdzero`. Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, `dmm.rtdalpha`, `dmm.rtdbeta`). If unique settings are desired, they must be changed, or use two different DMM configurations.

An error is generated if the value is out of range.

Example 1

```
dmm.func = "temperature"
dmm.transducer = dmm.TEMP_THREERTD
dmm.rtdalpha = 0.005
dmm.transducer = dmm.TEMP_FOURRTD
dmm.rtdalpha = 0.007
dmm.transducer = dmm.TEMP_THREERTD
print(dmm.rtdalpha)
```

Set an alpha constant for RTD to 0.005 for 3-wire RTD.

Change to 4-wire RTD and change the alpha constant to 0.007.

Switch back to 3-wire RTD. The value has been updated to 0.007.

Output:

`7.000000000e-03`

Example 2

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.rtdalpha = 0.005` `dmm.configure.set("RTD_3wire")` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.rtdalpha = 0.007` `dmm.configure.set("RTD_4wire")` `dmm.configure.recall("RTD_3wire")` `print(dmm.transducer, dmm.rtdalpha)` `dmm.configure.recall("RTD_4wire")` `print(dmm.transducer, dmm.rtdalpha)`	This example sets unique alpha constants for 3-wire and 4-wire RTDs by creating two DMM configurations with the desired settings. Output: `3.000000000e+00     5.000000000e-03` `4.000000000e+00     7.000000000e-03`

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.rtdbeta (on page 8-231)
dmm.rtddelta (on page 8-233)
dmm.rtdzero (on page 8-234)
dmm.transducer (on page 8-244)

dmm.rtdbeta

Indicates the user beta value for user type RTD.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.086300E-01

Usage

```
value = dmm.rtdbeta
dmm.rtdbeta = value
```

value	The user type RTD beta value; valid range is 0 to 1.0

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return nil when queried.

This setting only applies when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting is set but not used until the transducer type is set to an RTD type.

The RTD beta setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the RTD beta setting you set previously is retained.

NOTE

The following attributes share common settings and apply to both 3 and 4-wire RTDs: `dmm.rtdalpha`, `dmm.rtdbeta`, `dmm.rtddelta`, and `dmm.rtdzero`. Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, `dmm.rtdalpha`, `dmm.rtdbeta`). If unique settings are desired, they must be changed, or use two different DMM configurations.

An error is generated if the value is out of range.

Example 1

```
dmm.func = "temperature"
dmm.transducer = dmm.TEMP_THREERTD
dmm.rtdbeta = 0.3
dmm.transducer = dmm.TEMP_FOURRTD
dmm.rtdbeta = 0.5
dmm.transducer = dmm.TEMP_THREERTD
print(dmm.rtdbeta)
```

Set a beta constant for RTD to 0.3 for 3-wire RTD.
Change to 4-wire RTD.
Change the beta constant to 0.5.
Switch back to 3-wire RTD. The value is 0.5.
Output:

```
5.000000000e-01
```

Example 2

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.rtdbeta = 0.3` `dmm.configure.set("RTD_3wire")` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.rtdbeta = 0.5` `dmm.configure.set("RTD_4wire")` `dmm.configure.recall("RTD_3wire")` `print(dmm.transducer, dmm.rtdbeta)` `dmm.configure.recall("RTD_4wire")` `print(dmm.transducer, dmm.rtdbeta)`	This example sets unique beta constants for 3-wire and 4-wire RTDs by creating two DMM configurations with the desired settings. Output: `3.000000000e+00     3.000000000e-01` `4.000000000e+00     5.000000000e-01`

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.rtdalpha (on page 8-229)
dmm.rtddelta (on page 8-233)
dmm.rtdzero (on page 8-234)

dmm.rtddelta

The user type RTD delta value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.499900E+00

Usage

```
value = dmm.rtddelta
dmm.rtddelta = value
```

value	The user type RTD delta value; valid range is 0 to 5

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return nil when queried.

This setting only applies when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting is set but not used until the transducer type is set to an RTD type.

The RTD delta setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the RTD delta setting you set previously is retained.

NOTE

The following attributes share common settings and apply to both 3 and 4-wire RTDs: `dmm.rtdalpha`, `dmm.rtdbeta`, `dmm.rtddelta`, and `dmm.rtdzero`. Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, `dmm.rtdalpha`, `dmm.rtdbeta`). If unique settings are desired, they must be changed, or use two different DMM configurations.

An error is generated if the value is out of range.

Example 1

```
dmm.func = "temperature"
dmm.transducer = dmm.TEMP_THREERTD
dmm.rtddelta = 3
dmm.transducer = dmm.TEMP_FOURRTD
dmm.rtddelta = 5
dmm.transducer = dmm.TEMP_THREERTD
print(dmm.rtddelta)
```

Set a delta constant for RTD to 3 for 3-wire RTD.
Change to 4-wire RTD.
Change the delta constant to 5.
Switch back to 3-wire RTD. The value is 5.
Output:
`5.000000000e+00`

Example 2

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.rtddelta = 3` `dmm.configure.set("RTD_3wire")` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.rtddelta = 5` `dmm.configure.set("RTD_4wire")` `dmm.configure.recall("RTD_3wire")` `print(dmm.transducer, dmm.rtddelta)` `dmm.configure.recall("RTD_4wire")` `print(dmm.transducer, dmm.rtddelta)`	This example sets unique delta constants for 3-wire and 4-wire RTDs by creating two DMM configurations with the desired settings. Output: `3.000000000e+00 3.000000000e+00` `4.000000000e+00 5.000000000e+00`

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.rtdalpha (on page 8-229)
dmm.rtdbeta (on page 8-231)
dmm.rtdzero (on page 8-234)

dmm.rtdzero

Indicates the user type RTD zero value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+02

Usage

```
value = dmm.rtdzero
dmm.rtdzero = value
```

value	The user type RTD zero value; the range is 0 to 10000

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other configurations generate an error and return nil when queried.

This setting only applies when the transducer type is set to 3 or 4-wire RTD. For other transducer types, the setting is set but not used until the transducer type is set to an RTD type.

The RTD zero setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the RTD zero settings you set previously are retained.

NOTE

The following attributes share common settings and apply to both 3 and 4-wire RTDs: `dmm.rtdalpha`, `dmm.rtdbeta`, `dmm.rtddelta`, and `dmm.rtdzero`. Therefore, when both 3 and 4-wire RTDs are set to USER type for RTD, switching transducers between 3 and 4 will cause both to use the same settings (for example, `dmm.rtdalpha`, `dmm.rtdbeta`). If unique settings are desired, they must be changed, or use two different DMM configurations.

An error is generated if the value is out of range.

Example 1

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.rtdzero = 300` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.rtdzero = 500` `dmm.transducer = dmm.TEMP_THREERTD` `print(dmm.rtdzero)`	Set a zero constant for RTD to 300 for 3-wire RTD. Change to 4-wire RTD. Change the zero constant to 500. Switch back to 3-wire RTD. The value is 500. Output: `5.000000000e+02`

Example 2

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.rtdzero = 300` `dmm.configure.set("RTD_3wire")` `dmm.transducer = dmm.TEMP_FOURRTD` `dmm.rtdzero = 500` `dmm.configure.set("RTD_4wire")` `dmm.configure.recall("RTD_3wire")` `print(dmm.transducer, dmm.rtdzero)` `dmm.configure.recall("RTD_4wire")` `print(dmm.transducer, dmm.rtdzero)`	This example sets unique zero constants for 3-wire and 4-wire RTDs by creating two DMM configurations with the desired settings. Output: `3.000000000e+00     3.000000000e+02` `4.000000000e+00     5.000000000e+02`

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.rtdalpha (on page 8-229)
dmm.rtdbeta (on page 8-231)
dmm.rtddelta (on page 8-233)

dmm.savebuffer()

Saves data from the specified reading buffer to a USB flash drive using the specified filename.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dmm.savebuffer(bufferVar, fileName)
dmm.savebuffer(bufferVar, fileName, timeFormat)
```

bufferVar	A string that specifies the name of the DMM reading buffer that was created by `dmm.makebuffer()`
fileName	A string with the name of the file on the USB flash drive to which to save the DMM reading buffer
timeFormat	How date and time information from the buffer is saved in the file on the USB flash drive; the values are: • `dmm.buffer.SAVE_FORMAT_TIME`: The default. When this is selected, dates, times, and fractional seconds are saved • `dmm.buffer.SAVE_RELATIVE_TIME`: Relative time stamps are saved • `dmm.buffer.SAVE_RAW_TIME`: Seconds and fractional seconds are saved • `dmm.buffer.SAVE_TIMESTAMP_TIME`: Time stamps are saved

Details

The filename must specify the full path (including `/usb1/`). If included, the file extension must be set to .csv (if no file extension is specified, .csv is added).

For options that save more than one item of time information, each item is comma delimited. For example, the default format will be `<date>`, `<time>`, and `<fractional seconds>` for each reading, separated by commas.

You use `dmm.makebuffer()` to create a buffer.

Examples of valid destination file names:

```
dmm.savebuffer("bufferVar", "/usb1/myData")
dmm.savebuffer("bufferVar", "/usb1/myData.csv")
```

Invalid destination filename examples:

```
dmm.savebuffer("bufferVar", "/usb1/myData.")
```

— The period is not followed by the csv extension.

```
dmm.savebuffer("bufferVar", "/usb1/myData.txt")
```

— The only allowed extension is .csv. If .csv is not assigned, it is automatically added.

```
dmm.savebuffer("bufferVar", "/usb1/myData.txt.csv")
```

— Two periods in the file name (`myData_txt.csv` would be correct).

An error is generated if:

- The reading buffer does not exist or is not a DMM buffer.
- The destination filename is not specified correctly.
- The file extension is not .csv (or blank).

Example

`dmm.savebuffer("bufferVar", "/usb1/myData.csv")`	Saves readings from a DMM buffer named `bufferVar` with default time information to a file named `myData.csv` on the USB flash drive.
`dmm.savebuffer("bufferVar", "/usb1/myDataRel.csv", dmm.buffer.SAVE_RELATIVE_TIME)`	Saves readings from `bufferVar` with relative time stamps to a file named `myDataRel.csv` on the USB flash drive.

Also see

dmm.appendbuffer() (on page 8-152)
dmm.makebuffer() (on page 8-204)

dmm.setconfig()

Associates a DMM configuration with items specified in parameter channel list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Reset Channel reset Recall setup	Created configuration script Save setup	"nofunction"

Usage

```
dmm.setconfig(channelList, dmmConfiguration)
```

channelList	A string that lists the channels and channel patterns to change
dmmConfiguration	A string with the name of the DMM configuration that will be assigned to items in *channelList*

Details

dmmConfiguration can be the name of a configuration that was saved with `dmm.configure.set()`. If you use a saved configuration, the function of the configuration and the supporting DMM attributes for that function are associated with the *channelList* parameter items. These supporting DMM attributes may have user-defined or default values associated with them.

dmmConfiguration can also be a DMM configuration name that matches the DMM function name. If you use a default DMM configuration name, be aware that the supporting function attribute settings are the default values and not user-specified (as they may be in a user-defined saved configuration). The DMM function names are:

- `"accurrent"`
- `"acvolts"`
- `"commonsideohms"`
- `"continuity"`
- `"dccurrent"`
- `"dcvolts"`
- `"fourwireohms"`
- `"frequency"`
- `"nofunction"`
- `"period"`
- `"temperature"`
- `"twowireohms"`

To use a channel with the `dmm.close()` function, `dmm.setconfig()` cannot be set to `"nofunction"`.

The configuration being assigned determines whether analog backplane relay 1 or 2 get used, based on the function associated with the configuration when being assigned to a channel. For channel patterns, the pattern image must include the desired analog backplane relays along with the desired channels. This command has no effect on the poles setting for a channel (`channel.setpole()`) or analog backplane relays specified by `channel.setbackplane()` function.

.An error is generated if:

- There is more than one DMM configuration specified.
- A DMM configuration is specified that does not exist.
- The desired DMM functionality is not supported on a specified channel.
- An analog backplane relay is specified.
- A specified channel does not exist for the card installed on the slot specified.
- A specified channel is forbidden to close.
- A matrix channel is in channel list parameter (for example, the Model 3730 is 6 x 16 high density matrix card, so an error is generated if a Model 3730 channel is included in the channel list parameter).

Once an error is detected, the command stops processing and no channels or channel patterns are modified.

Example

`dmm.setconfig("1001:3100", "myDcv")`	Assigns `myDcv` to all the channels on slots 1 and 2 and channels 1 to 100 on slot 3.
`dmm.setconfig("slot5", "dcvolts")`	Assigns the factory default settings for `dcvolts` to channels on slot 5.

Also see

channel.setbackplane() (on page 8-90)
channel.setpole() (on page 8-101)
dmm.close() (on page 8-167)
dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.getconfig() (on page 8-189)

dmm.simreftemperature

The simulated reference temperature for thermocouples.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	2.300000E+01 (23°C)

Usage

```
value = dmm.simreftemperature
dmm.simreftemperature = value
```

value	The simulated reference temperature in Celsius (0 °C to 65 °C), Fahrenheit (32 °F to 149 °F), or Kelvin (273 °K to 338 °K)

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return `nil` when queried.

The simulated reference temperature is only used when the transducer type is thermocouple, as set by `dmm.transducer`. For all other transducer types, the value is set but not used until the transducer type is set for thermocouple.

The simulated reference temperature setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"` with the transducer type set to thermocouple, the simulated reference temperature setting you set previously is retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMOCOUPLE` `dmm.units = dmm.UNITS_CELSIUS` `dmm.simreftemperature = 30`	Sets 30 degrees Celsius as the simulated reference temperature for thermocouples.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)
dmm.units (on page 8-245)

dmm.thermistor

The type of thermistor.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	5000

Usage

```
value = dmm.thermistor
dmm.thermistor = value
```

value	The thermistor type in ohms, 2252, 5000 or 10000; if you enter any other value, it is converted as shown in the following table:

Parameter	Converted value
>= 1950 and < 3500	2252
>= 3500 and < 7500	5000
>= 7500 and <= 10050	10000

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return `nil` when queried. If you use a parameter outside of the ranges listed in the usage table, a parameter out of range error message is generated.

The thermistor attribute is only used when the transducer type is set for thermistor. For all other transducer types, the setting is set but not used until thermistor is selected for the transducer type (see `dmm.transducer`).

The thermistor setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the thermistor setting you set previously is retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMISTOR` `dmm.thermistor = 3000`	Sets thermistor type to 2252. Note that the original value is set to 3000, but is automatically converted to 2252.
`print(dmm.thermistor)`	2252

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)

dmm.thermocouple

Indicates the thermocouple type.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.THERMOCOUPLE_K)

Usage

```
value = dmm.thermocouple
dmm.thermocouple = value
```

value	The thermocouple type: • dmm.THERMOCOUPLE_J or 0 • dmm.THERMOCOUPLE_K or 1 • dmm.THERMOCOUPLE_T or 2 • dmm.THERMOCOUPLE_E or 3 • dmm.THERMOCOUPLE_R or 4 • dmm.THERMOCOUPLE_S or 5 • dmm.THERMOCOUPLE_B or 6 • dmm.THERMOCOUPLE_N or 7

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return nil when queried. An illegal parameter value error message is generated if the value specified is not a supported thermocouple type value listed in the usage table.

The thermocouple attribute is only used when the transducer type is thermocouple (see `dmm.transducer`). For all other transducer types, the value is set but not used until the transducer type is set for thermocouple.

The thermocouple setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the thermocouple value you set previously is retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMOCOUPLE` `dmm.thermocouple = dmm.THERMOCOUPLE_J`	Sets the thermocouple type to J.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)

dmm.threertd

The type of three-wire RTD being used.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.RTD_PT100)

Usage

```
value = dmm.threertd
dmm.threertd = value
```

value	The desired type for 3-wire RTD: • dmm.RTD_PT100 or 0 for type PT100 • dmm.RTD_D100 or 1 for type D100 • dmm.RTD_F100 or 2 for type F100 • dmm.RTD_PT385 or 3 for type PT385 • dmm.RTD_PT3916 or 4 for type PT3916 • dmm.RTD_USER or 5 for user-specified type

Details

This attribute is only valid when `dmm.func` is set to `"temperature"` and `dmm.transducer` is set to `dmm.TEMP_THREERTD`. For all other transducer types, the attribute is set but is not used until the transducer type is set for three-wire RTD. All other functions generate an error and return nil when queried.

An illegal parameter value error message is generated if the value specified is not a supported RTD type value as listed in the usage table.

The three-wire RTD setting is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the three-wire RTD setting you set previously is retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THREERTD` `dmm.threertd = dmm.RTD_PT3916`	Sets the type of three-wire RTD to PT3916.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)
dmm.transducer (on page 8-244)

dmm.threshold

Indicates the threshold range.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1.000000E+01

Usage

```
value = dmm.threshold
dmm.threshold = value
```

value	The desired threshold setting. The range for: • Continuity is from 1 to 1000 Ω • Frequency and period is from 0 to 303 V

Details

This attribute is only valid when `dmm.func` is set to `"frequency"`, `"period"`, or `"continuity"`. All other functions generate an error and return nil when queried.

For frequency and period, this refers to a threshold voltage range.

For continuity, it refers to a threshold resistance in ohms.

Errors are generated if the parameter value does not make sense for selected function.

The threshold value is saved with the `dmm.func` function setting, so if you use another function, then return to `"frequency"`, `"period"`, or `"continuity"`, the threshold value you set previously is retained.

Example

`dmm.func = "frequency"` `dmm.threshold = 30`	Sets the threshold range for frequency to 30.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

dmm.transducer

The transducer type.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	1 (dmm.TEMP_THERMOCOUPLE)

Usage

```
value = dmm.transducer
dmm.transducer = value
```

value	The transducer type: • dmm.TEMP_THERMOCOUPLE or 1 • dmm.TEMP_THERMISTOR or 2 • dmm.TEMP_THREERTD or 3 • dmm.TEMP_FOURRTD or 4

Details

This attribute is only valid when `dmm.func` is set to `"temperature"`. All other functions generate an error and return nil when queried.

NOTE

The setting of this attribute affects which other temperature-supported attributes get used. There are various attributes that are only applicable when the transducer type is a certain type. Although the transducer type needs to match for the attribute setting to be used, the transducer type does not need to match to change the setting or read the setting. For example, the transducer type does not need to be set to `dmm.TEMP_FOURRTD` to change the `dmm.fourrtd` attribute setting.

The transducer value is saved with the `dmm.func` function setting, so if you use another function, then return to `"temperature"`, the transducer value you set previously is retained.

Example

`dmm.func = "temperature"` `dmm.transducer = dmm.TEMP_THERMISTOR`	Sets transducer to thermistor type.

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.fourrtd (on page 8-186)
dmm.func (on page 8-187)

dmm.units

The units that are used for voltage and temperature measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset DMM reset Recall setup	Create configuration script Save setup	0 (dmm.UNITS_VOLTS) for "acvolts" and "dcvolts" 2 (dmm.UNITS_CELSIUS) for "temperature"

Usage

```
state = dmm.units
dmm.units = state
```

value	For `dcvolts` and `acvolts`, select from the following units: • dmm.UNITS_VOLTS or 0 • dmm.UNITS_DECIBELS or 1 For `temperature`, select from the following units: • dmm.UNITS_CELSIUS or 2 • dmm.UNITS_KELVIN or 3 • dmm.UNITS_FAHRENHEIT or 4

Details

This attribute is only valid when `dmm.func` is set to `"dcvolts"`, `"acvolts"`, or `"temperature"`.

All other functions generate an error and return nil when queried.

The units value is saved with the `dmm.func` function setting, so if you use another function, then return to `"dcvolts"`, `"acvolts"`, or `"temperature"`, the units setting you set previously is retained.

Errors are generated if the parameter value does not make sense for the selected function.

Example

`dmm.func = "temperature"` `dmm.units = dmm.UNITS_FAHRENHEIT`	Sets units for temperature measurements to Fahrenheit (°F).

Also see

dmm.configure.recall() (on page 8-173)
dmm.configure.set() (on page 8-175)
dmm.func (on page 8-187)

errorqueue.clear()

This function clears all entries out of the error queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
errorqueue.clear()
```

Details

See the Error queue topic for additional information about the error queue.

Also see

Error queue
errorqueue.count (on page 8-246)
errorqueue.next() (on page 8-246)

errorqueue.count

This attribute gets the number of entries in the error queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Clearing error queue Reading error messages	Not applicable	Not applicable

Usage

```
count = errorqueue.count
```

count	The number of entries in the error queue

Example

`count = errorqueue.count` `print(count)`	Returns the number of entries in the error queue. The output below indicates that there are four entries in the error queue: `4.00000e+00`

Also see

errorqueue.clear() (on page 8-246)
errorqueue.next() (on page 8-246)

errorqueue.next()

This function reads the oldest entry from the error queue and removes it from the queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`errorCode, message, severity, errorNode = errorqueue.next()`

errorCode	The error code number for the entry
message	The message that describes the error code
severity	The severity level (0, 10, 20, 30, or 40); see **Details** for more information
errorNode	The node number where the error originated

Details

Entries are stored in a first-in, first-out (FIFO) queue. This functions reads the oldest entry and removes it from the queue.

Error codes and messages are listed in the Error summary list.

If there are no entries in the queue, code 0, "Queue is Empty" is returned.

Returned severity levels are described in the following table.

Severity level descriptions		
Number	**Level**	**Description**
0	**Informational**	Indicates that there are no entries in the queue.
10	**Informational**	Indicates a status message or minor error.
20	**Recoverable**	Indicates possible invalid user input; operation continues but action should be taken to correct the error.
30	**Serious**	Indicates a serious error that may require technical assistance, such as corrupted data.
40	**Fatal**	Instrument is not operational.

In an expanded system, each TSP-Link enabled instrument is assigned a node number. The variable *errorNode* stores the node number where the error originated.

Example

`errorcode, message = errorqueue.next()` `print(errorcode, message)`	Reads the oldest entry in the error queue. The output below indicates that the queue is empty. Output: `0.00000e+00 Queue Is Empty`

Also see

Error queue
errorqueue.clear() (on page 8-246)
errorqueue.count (on page 8-246)
Error summary list

eventlog.all()

This function returns all entries from the event log as a single string and removes them from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`logString = eventlog.all()`

logString	A listing of all event log entries

Details

This function returns all events in the event log. Logged items are shown from oldest to newest. The response is a string that has the messages delimited with a new line character.

This function also clears the event log.

If there are no entries in the event log, this function returns the value `nil`.

Example

```
print(eventlog.all())
```

Get and print all entries from the event log and remove the entries from the log.

Output:

```
17:26:35.690 10 Oct 2007, LAN0, 192.168.1.102, LXI, 0, 1192037132,
   1192037155.733269000, 0, 0x0
17:26:39.009 10 Oct 2007, LAN5, 192.168.1.102, LXI, 0, 1192037133,
   1192037159.052777000, 0, 0x0
```

Also see

eventlog.clear() (on page 8-248)
eventlog.count (on page 8-249)
eventlog.enable (on page 8-249)
eventlog.next() (on page 8-250)
eventlog.overwritemethod (on page 8-251)

eventlog.clear()

This command clears the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Details

This function removes all messages from the event log.

Also see

eventlog.all() (on page 8-247)
eventlog.count (on page 8-249)
eventlog.enable (on page 8-249)
eventlog.next() (on page 8-250)
eventlog.overwritemethod (on page 8-251)

eventlog.count

This attribute returns the number of events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Clearing event log Reading event log	Not applicable	Not applicable

Usage

N = `eventlog.count`

N	The number of events in the event log

Example

`print(eventlog.count)`	Displays the present number of events in the instrument event log. Output looks similar to: `3.00000e+00`

Also see

eventlog.all() (on page 8-247)
eventlog.clear() (on page 8-248)
eventlog.enable (on page 8-249)
eventlog.next() (on page 8-250)
eventlog.overwritemethod (on page 8-251)

eventlog.enable

This attribute enables or disables the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	eventlog.ENABLE

Usage

```
status = eventlog.enable
eventlog.enable = status
```

status	The enable status of the event log: `1` or `eventlog.ENABLE`: Event log enable `0` or `eventlog.DISABLE`: Event log disable

Details

When the event log is disabled (`eventlog.DISABLE` or `0`), no new events are added to the event log. You can, however, read and remove existing events.

When the event log is enabled, new events are logged.

Example

`print(eventlog.enable)` `eventlog.enable = eventlog.DISABLE` `print(eventlog.enable)`	Displays the present status of the Series 3700A event log. Output: `1.00000e+00` `0.00000e+00`

Also see

eventlog.all() (on page 8-247)
eventlog.clear() (on page 8-248)
eventlog.count (on page 8-249)
eventlog.next() (on page 8-250)
eventlog.overwritemethod (on page 8-251)

eventlog.next()

This function returns the oldest message from the event log and removes it from the log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
logString = eventlog.next()
```

logString	The next log entry

Details

Returns the next entry from the event log and removes it from the log.

If there are no entries in the event log, returns the value `nil`.

Example 1

```
print(eventlog.next())
```

Get the oldest message in the event log and remove that entry from the log.

Output:

```
17:28:22.085 10 Oct 2009, LAN2, 192.168.1.102, LXI, 0, 1192037134, <no time>, 0,
    0x0
```

Example 2

```
print(eventlog.next())
```

If you send this command when there is nothing in the event log, you will get the following output:

```
nil
```

Also see

eventlog.all() (on page 8-247)
eventlog.clear() (on page 8-248)
eventlog.count (on page 8-249)
eventlog.enable (on page 8-249)
eventlog.overwritemethod (on page 8-251)

eventlog.overwritemethod

This attribute controls how the event log processes events if the event log is full.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Save setup	1 (eventlog.DISCARD_OLDEST)

Usage

```
method = eventlog.overwritemethod
eventlog.overwritemethod = method
```

method	Set to one of the following values: • `0` or `eventlog.DISCARD_NEWEST`: New entries are not logged • `1` or `eventlog.DISCARD_OLDEST`: Old entries are deleted as new events are logged

Details

When this attribute is set to `eventlog.DISCARD_NEWEST`, new entries are not logged.

When this attribute is set to `eventlog.DISCARD_OLDEST`, the oldest entry is discarded when a new entry is added.

Example

`eventlog.overwritemethod = 0`	When the log is full, the event log will ignore new entries.

Also see

eventlog.all() (on page 8-247)
eventlog.clear() (on page 8-248)
eventlog.count (on page 8-249)
eventlog.enable (on page 8-249)
eventlog.next() (on page 8-250)

exit()

This function stops a script that is presently running.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
exit()
```

Details

Terminates script execution when called from a script that is being executed.

This command does not wait for overlapped commands to complete before terminating script execution. If overlapped commands are required to finish, use the `waitcomplete()` function before calling `exit()`.

Also see

waitcomplete() (on page 8-466)

fileVar:close()

This function closes the file that is represented by the *fileVar* variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

fileVar:close()

fileVar	The file descriptor variable to close

Details

This command is equivalent to io.close(*fileVar*).

Note that files are automatically closed when the file descriptors are garbage collected.

Also see

File I/O (on page 3-25)
fileVar:flush() (on page 8-252)
fileVar:read() (on page 8-253)
fileVar:seek() (on page 8-254)
fileVar:write() (on page 8-254)
io.close() (on page 8-262)
io.open() (on page 8-264)

fileVar:flush()

This function writes buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

fileVar:flush()

fileVar	The file descriptor variable to flush

Details

The *fileVar*:write() or io.write() functions buffer data, which may not be written immediately to the USB flash drive. Use *fileVar*:flush() to flush this data. Using this function removes the need to close a file after writing to it, allowing the file to be left open to write more data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, and you want to keep the file open, flush the file after you write to it to prevent loss of data.

Also see

File I/O (on page 3-25)
fileVar:write() (on page 8-254)
io.open() (on page 8-264)
io.write() (on page 8-266)

fileVar:read()

This function reads data from a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
data1 = fileVar:read()
data1 = fileVar:read(format1)
data1, data2 = fileVar:read(format1, format2)
data1, ..., datan = fileVar:read(format1, ..., formatn)
```

data1	First data read from the file
data2	Second data read from the file
datan	Last data read from the file
fileVar	The descriptor of the file to be read
format1	A string or number indicating the first type of data to be read
format2	A string or number indicating the second type of data to be read
formatn	A string or number indicating the last type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

`"*n"`: Returns a number.

`"*a"`: Returns the whole file, starting at the current position (returns an empty string if the current file position is at the end of the file).

`"*l"`: Returns the next line, skipping the end of line; returns `nil` if the current file position is at the end of file.

n: Returns a string with up to n characters; returns an empty string if n is zero; returns `nil` if the current file position is at the end of file.

If no format parameters are provided, the function will perform as if the function is passed the value `"*l"`.

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Also see

File I/O (on page 3-25)
fileVar:write() (on page 8-254)
io.input() (on page 8-263)
io.open() (on page 8-264)

fileVar:seek()

This function sets and gets a file's current position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
position, errorMsg = fileVar:seek()
position, errorMsg = fileVar:seek(whence)
position, errorMsg = fileVar:seek(whence, offset)
```

position	The new file position, measured in bytes from the beginning of the file
errorMsg	A string containing the error message
fileVar	The file descriptor variable
whence	A string indicating the base against which *offset* is applied; the default is `"cur"`
offset	The intended new position, measured in bytes from a base indicated by *whence* (default is `0`)

Details

The *whence* parameters may be any of the following:

`"set"`: Beginning of file

`"cur"`: Current position

`"end"`: End of file

If an error is encountered, it is logged to the error queue, and the command returns `nil` and the error string.

Also see

File I/O (on page 3-25)
io.open() (on page 8-264)

fileVar:write()

This function writes data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar:write(data)
fileVar:write(data1, data2)
fileVar:write(data1, ..., datan)
```

fileVar	The file descriptor variable
data	Write all data to the file
data1	The first data to write to the file
data2	The second data to write to the file
datan	The last data to write to the file
...	One or more entries (or values) separated by commas

Details

This function may buffer data until a flush (`fileVar:flush()` or `io.flush()`) or close (`fileVar:close()` or `io.close()`) operation is performed.

Also see

File I/O (on page 3-25)
fileVar:close() (on page 8-252)
fileVar:flush() (on page 8-252)
io.close() (on page 8-262)
io.flush() (on page 8-262)
io.open() (on page 8-264)

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	Instrument reset Recall setup	Create configuration script Save setup	***need to define ASCIIPrecision variable***

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions; must be a number between 1 and 16

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format. The precision value must be a number between 1 and 16.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

`format.asciiprecision = 10` `x = 2.54` `printnumber(x)` `format.asciiprecision = 3` `printnumber(x)`	Output: `2.540000000e+00` `2.54e+00`

Also see

format.byteorder (on page 8-256)
format.data (on page 8-257)
print() (on page 8-306)
printbuffer() (on page 8-307)
printnumber() (on page 8-310)

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)	Yes	Instrument reset	Create configuration script	format.LITTLEENDIAN

Usage

```
order = format.byteorder
format.byteorder = order
```

order	Byte order value as follows: • Most significant byte first: `0`, `format.NORMAL`, `format.NETWORK`, or `format.BIGENDIAN` • Least significant byte first: `1`, `format.SWAPPED` or `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.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` data formats.

`format.NORMAL`, `format.BIGENDIAN`, and `format.NETWORK` select the same byte order. `format.SWAPPED` and `format.LITTLEENDIAN` select the same byte order. Selecting which to use is a matter of preference.

Select the `format.SWAPPED` or `format.LITTLEENDIAN` byte order when sending data to a computer with a Microsoft Windows operating system.

Example

```
x = 1.23
format.data = format.REAL32
format.byteorder = format.LITTLEENDIAN
printnumber(x)
format.byteorder = format.BIGENDIAN
printnumber(x)
```

Output depends on the terminal program you use, but will look something like:

```
#0¤p??
#0??p¤
```

Also see

format.asciiprecision (on page 8-255)
format.data (on page 8-257)
printbuffer() (on page 8-307)
printnumber() (on page 8-310)

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	Instrument reset Recall setup	Create configuration script Saved setup	1 (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: `1` or `format.ASCII` • Single-precision IEEE Std 754 binary format: 2, `format.SREAL`, or `format.REAL32` • Double-precision IEEE Std 754 binary format: 3, `format.REAL`, `format.REAL64`, or `format.DREAL`

Details

The precision of numeric values can be controlled with the `format.asciiprecision` attribute. The byte order of `format.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` can be selected with the `format.byteorder` attribute.

REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double precision format. They are alternative identifiers. Selecting which to use is a matter of preference.

The IEEE 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.

NOTE

Binary formats are not intended to be interpreted by humans.

Example

`format.asciiprecision = 10` `x = 3.14159265` `format.data = format.ASCII` `printnumber(x)` `format.data = format.REAL64` `printnumber(x)`	Output a number represented by *x* in ASCII using a precision of 10, then output the same number in binary using double precision format. Output: `3.141592650e+00` `#0ñÔÈSû!    @`

Also see

format.asciiprecision (on page 8-255)
format.byteorder (on page 8-256)
printbuffer() (on page 8-307)
printnumber() (on page 8-310)

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)`

workingDirectory	Returned value containing the working path
path	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

`testPath = fs.chdir("/usb1/")`	Change the working directory to `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

Also see

File I/O (on page 3-25)

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.

An error is logged to the error queue if the given path does not exist.

Also see

File I/O (on page 3-25)
fs.is_file() (on page 8-259)

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.

An error is logged to the error queue if the given path does not exist.

Also see

File I/O (on page 3-25)
fs.is_dir() (on page 8-258)

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.

Also see

File I/O (on page 3-25)
fs.rmdir() (on page 8-260)

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.

Also see

File I/O (on page 3-25)

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)
```

path	The path of the directory to remove

Details

This path may be absolute or relative to the present current directory.

An error is logged to the error queue if the given path does not exist, or does not represent a directory, or if the directory is not empty.

Also see

File I/O (on page 3-25)
fs.mkdir() (on page 8-259)

gettimezone()

This function retrieves the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`timeZone = gettimezone()`

timeZone	The local timezone of the instrument

Details

See `settimezone()` for additional details about the time zone format and a description of the fields.

timeZone can be in either of the following formats:

- If one parameter was used with `settimezone()`, the format used is:
  `GMThh:mm:ss`
- If four parameters were used with `settimezone()`, the format used is:
  `GMThh:mm:ssGMThh:mm:ss,Mmm.w.dw/hh:mm:ss,Mmm.w.dw/hh:mm:ss`

Example

`timezone = gettimezone()`	Reads the value of the local timezone.

Also see

settimezone() (on page 8-367)

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
```

address	The GPIB address of the instrument (0 to 30)

Details

The GPIB address value is set to 16 at the factory. The address can be set to any address value between 0 and 30. However, the address must be unique in the system. It cannot conflict with an address that is assigned to another instrument or to the GPIB controller.

A new GPIB address takes effect when the command to change it is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the GPIB interface.

You should allow ample time for the command to be processed before attempting to communicate with the instrument again. After sending this command, make sure to use the new address to communicate with the instrument.

The `reset()` function does not affect the GPIB address.

Example

`gpib.address = 26` `address = gpib.address` `print(address)`	Sets the GPIB address and reads the address. Output: `2.600000e+01`

Also see

GPIB setup (on page 2-58)

io.close()

This function closes a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
io.close()
io.close(file)
```

file	The descriptor of the file to close

Details

If a file is not specified, the default output file closes.

Only `io.close()`, used without specifying a parameter, can be accessed from a remote node.

Example

`testFile, testError = io.open("testfile.txt", "w")` `if nil == testError then` `   testFile:write("This is my test file")` `   io.close(testFile)` `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

fileVar:close (see "fileVar:close()" on page 8-252)
Script examples (on page 3-27)
io.open() (on page 8-264)

io.flush()

This function saves buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
io.flush()
```

Details

You must use the `io.flush()` or `io.close()` functions to write data to the file system.

NOTE

Data is not automatically written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

This function only flushes the default output file.

Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written (and when you want to keep the file open and not close it), flush the file after writing to it.

Also see

Script examples (on page 3-27)
fileVar:flush() (on page 8-252)
fileVar:write() (on page 8-254)
io.write() (on page 8-266)

io.input()

This function assigns a previously opened file, or opens a new file, as the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.input()
fileVar = io.input(newfile)
```

fileVar	The descriptor of the input file or an error message (if the function fails)
newfile	A string representing the path of a file to open as the default input file, or the file descriptor of an open file to use as the default input file

Details

The *newfile* path may be absolute or relative to the current working directory.

When using this function from a remote TSP-Link® node, this command does not accept a file descriptor and does not return a value.

If the function fails, an error message is returned.

Also see

Script examples (on page 3-27)
io.open() (on page 8-264)
io.output() (on page 8-264)

io.open()

This function opens a file for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar, errorMsg = io.open(path)
fileVar, errorMsg = io.open(path, mode)
```

fileVar	The descriptor of the opened file
errorMsg	Indicates whether an error was encountered while processing the function
path	The path of the file to open
mode	A string representing the intended access mode (`"r"` = read, `"w"` = write, and `"a"` = append)

Details

The path to the file to open may be absolute or relative to the current working directory. If you successfully open the file, *errorMsg* is nil and *fileVar* has the descriptor that can be used to access the file.

If an error is encountered, the command returns `nil` for *fileVar* and an error string.

Example

`testFile, testError = io.open("testfile.txt", "w")` `if testError == nil then` `   testFile:write("This is my test file")` `   io.close(testFile)` `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

Script examples (on page 3-27)
fileVar:close (see "fileVar:close()" on page 8-252)
io.close() (on page 8-262)

io.output()

This function assigns a previously opened file or opens a new file as the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.output()
fileVar = io.output(newfile)
```

fileVar	The descriptor of the output file or an error message (if the function fails)
newfile	A file descriptor to assign (or the path of a file to open) as the default output file

Details

The path of the file to open may be absolute or relative to the current working directory.

When accessed from a remote node using the TSP-Link network, this command does not accept a file descriptor parameter and does not return a value.

If the function fails, an error message is returned.

Also see

Script examples (on page 3-27)
io.input() (on page 8-263)
io.open() (on page 8-264)

io.read()

This function reads data from the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data1 = io.read()
data1 = io.read(format1)
data1, data2 = io.read(format1, format2)
data1, ..., dataN = io.read(format1, ..., formatN)
```

data1	The data read from the file
data2	The data read from the file
dataN	The data read from the file; the number of return values matches the number of format values given
format1	A string or number indicating the type of data to be read
format2	A string or number indicating the type of data to be read
formatN	A string or number indicating the type of data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

Format parameter	Description
`"*N"`	Returns a number
`"*a"`	Returns the whole file, starting at the present position; returns an empty string if it is at the end of file
`"*l"`	Returns the next line, skipping the end of line; returns `nil` if the present file position is at the end of file
N	Returns a string with up to *N* characters; returns an empty string if *N* is zero (0); returns `nil` if the present file position is at the end of file

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

If no format parameters are provided, the function will perform as if the function was passed the value `"*l"`.

Also see

fileVar:read() (on page 8-253)
Script examples (on page 3-27)

io.type()

This function checks whether or not a given object is a file handle.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = io.type(obj)
```

type	Indicates whether the object is an open file handle
obj	Object to check

Details

Returns the string `"file"` if the object is an open file handle. If it is not an open file handle, `nil` is returned.

Also see

Script examples (on page 3-27)
io.open() (on page 8-264)

io.write()

This function writes data to the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
io.write()
io.write(data1)
io.write(data1, data2)
io.write(data1, ..., dataN)
```

data1	The data to be written
data2	The data to be written
dataN	The data to be written
...	One or more values separated by commas

Details

All data parameters must be either strings or numbers.

NOTE

Data is not immediately written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

Also see

Script examples (on page 3-27)
io.flush() (on page 8-262)

lan.applysettings()

This function re-initializes the LAN interface with new settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.applysettings()
```

Details

Disconnects all existing LAN connections to the instrument and re-initializes the LAN with the present configuration settings.

This function initiates a background operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization continues to run in the background.

Even though the LAN configuration settings may not have changed since the LAN was last connected, new settings may take effect due to the dynamic nature of dynamic host configuration protocol (DHCP) or dynamic link local addressing (DLLA) configuration.

Re-initialization takes effect even if the configuration has not changed since the last time the instrument connected to the LAN.

Example

`lan.applysettings()`	Re-initialize the LAN interface with new settings.

Also see

None

lan.config.dns.address[N]

Configures DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
dnsAddress = lan.config.dns.address[N]
lan.config.dns.address[N] = dnsAddress
```

dnsAddress	DNS server IP address
N	Entry index (1 or 2)

Details

This attribute is an array of DNS (domain name system) server addresses. These addresses take priority for DNS lookups and are consulted before any server addresses that are obtained using DHCP. This allows local DNS servers to be specified that take priority over DHCP-configured global DNS servers.

You can specify up to two addresses. The address specified by 1 is consulted first for DNS lookups. *dnsAddress* must be a string specifying the DNS server's IP address in dotted decimal notation.

Unused entries are returned as `"0.0.0.0"` when read. To disable an entry, set its value to `"0.0.0.0"` or the empty string `""`.

Although only two address may be manually specified here, the instrument will use up to three DNS server addresses. If two are specified here, only one that is given by a DHCP server is used. If no entries are specified here, up to three addresses that are given by a DHCP server are used.

Example

`dnsaddress = "164.109.48.173"` `lan.config.dns.address[1] = dnsaddress`	Configure DNS address 1 to "164.109.48.173"

Also see

lan.config.dns.domain (on page 8-268)
lan.config.dns.dynamic (on page 8-269)
lan.config.dns.hostname (on page 8-269)
lan.config.dns.verify (on page 8-270)
lan.restoredefaults() (on page 8-274)

lan.config.dns.domain

Configures the dynamic DNS domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	""

Usage

```
domain = lan.config.dns.domain
lan.config.dns.domain = domain
```

domain	Dynamic DNS registration domain; use a string of 255 characters or less

Details

This attribute holds the domain to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the domain specified in this attribute with the DNS server.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 255 characters are allowed, you must make sure the combined length is also no more than 255 characters.

Example

`print(lan.config.dns.domain)`	Outputs the present dynamic DNS domain. For example, if the domain is "Matrix", the response would be: `Matrix`

Also see

lan.config.dns.dynamic (on page 8-269)
lan.config.dns.hostname (on page 8-269)
lan.config.dns.verify (on page 8-270)
lan.restoredefaults() (on page 8-274)

lan.config.dns.dynamic

Enables or disables the dynamic DNS registration.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.dynamic
lan.config.dns.dynamic = state
```

state	The dynamic DNS registration state. It may be one of the following values: `1` or `lan.ENABLE`: Enabled `0` or `lan.DISABLE`: Disabled

Details

Dynamic DNS registration works with DHCP to register the host name with the DNS server. The host name is specified in the `lan.config.dns.hostname` attribute.

Example

`print(lan.config.dns.dynamic)`	Outputs the dynamic registration state. If dynamic DNS registration is enabled, the response is: `1.00000e+00`

Also see

lan.config.dns.hostname (on page 8-269)
lan.restoredefaults() (on page 8-274)

lan.config.dns.hostname

This attribute defines the dynamic DNS host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see **Details**)

Usage

```
hostName = lan.config.dns.hostname
lan.config.dns.hostname = hostName
```

hostName	The host name to use for dynamic DNS registration; the host name must: • be a string of 15 characters or less • start with a letter • end with a letter or digit • contain only letters, digits, and hyphens

Details

This attribute holds the host name to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the host name specified in this attribute with the DNS server.

The factory default value for `hostName` is `"K-<model number>-<serial number>"`, where `<model number>` and `<serial number>` are replaced with the actual model number and serial number of the instrument (for example, `"K-3706A-1234567"`). Note that hyphens separate the characters of `hostName`.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 15 characters can be entered here, care must be taken to be sure the combined length is no more than 255 characters.

Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero, or one consisting entirely of whitespace characters) will revert the host name to the factory default value.

Example

`print(lan.config.dns.hostname)`	Outputs the present dynamic DNS host name.

Also see

lan.config.dns.dynamic (on page 8-269)
lan.restoredefaults() (on page 8-274)

lan.config.dns.verify

This attribute defines the DNS host name verification state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	1 (lan.ENABLE)

Usage

```
state = lan.config.dns.verify
lan.config.dns.verify = state
```

state	DNS hostname verification state: `1` or `lan.ENABLE`: DNS host name verification enabled `0` or `lan.DISABLE`: DNS host name verification disabled

Details

When this is enabled, the instrument performs DNS lookups to verify that the DNS host name matches the value specified by `lan.config.dns.hostname`.

Example

`print(lan.config.dns.verify)`	Outputs the present DNS host name verification state. If it is enabled, the output is: `1.00000e+00`

Also see

lan.config.dns.hostname (on page 8-269)
lan.restoredefaults() (on page 8-274)

lan.config.gateway

This attribute contains the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
gatewayAddress = lan.config.gateway
lan.config.gateway = gatewayAddress
```

gatewayAddress	LAN default gateway address; must be a string specifying the default gateway's IP address in dotted decimal notation

Details

This attribute specifies the default gateway IP address to use when manual or DLLA configuration methods are used to configure the LAN. If DHCP is enabled, this setting is ignored.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.gateway` attribute to determine the present operating state of the LAN.

The IP address must be formatted in four groups of numbers, each separated by a decimal.

Example

`print(lan.config.gateway)`	Outputs the default gateway address. For example, you might see the output: `192.168.0.1`

Also see

lan.restoredefaults() (on page 8-274)
lan.status.gateway (on page 8-277)

lan.config.ipaddress

This command specifies the LAN IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"192.168.0.2"

Usage

```
ipAddress = lan.config.ipaddress
lan.config.ipaddress = ipAddress
```

ipAddress	LAN IP address; must be a string specifying the IP address in dotted decimal notation

Details

This command specifies the LAN IP address to use when the LAN is configured using the manual configuration method. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.ipaddress` attribute to determine the present operating state of the LAN.

Example

`ipaddress = lan.config.ipaddress`	Retrieves the presently set LAN IP address.

Also see

lan.restoredefaults() (on page 8-274)
lan.status.ipaddress (on page 8-277)

lan.config.method

This attribute contains the LAN settings configuration method.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0 (lan.AUTO)

Usage

```
method = lan.config.method
lan.config.method = method
```

method	The method for configuring LAN settings; it can be one of the following values: `0` or `lan.AUTO`: Selects automatic sequencing of configuration methods `1` or `lan.MANUAL`: Use only manually specified configuration settings

Details

This attribute controls how the LAN IP address, subnet mask, default gateway address, and DNS server addresses are determined.

When method is `lan.AUTO`, the instrument 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, it uses the manually specified settings.

When method is `lan.MANUAL`, only the manually specified settings are used. Neither DHCP nor DLLA are attempted.

Example

`print(lan.config.method)`	Outputs the current method. For example: `1.00000e+00`

Also see

lan.restoredefaults() (on page 8-274)

lan.config.subnetmask

This attribute contains the LAN subnet mask.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"255.255.255.0"

Usage

```
mask = lan.config.subnetmask
lan.config.subnetmask = mask
```

mask	String that specifies the LAN subnet mask value in dotted decimal notation

Details

This attribute specifies the LAN subnet mask that will be used when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.subnetmask` attribute to determine the present operating state of the LAN.

Example

`print(lan.config.subnetmask)`	Outputs the LAN subnet mask, such as: `255.255.255.0`

Also see

lan.restoredefaults() (on page 8-274)
lan.status.subnetmask (on page 8-281)

lan.lxidomain

This attribute contains the LXI domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	0

Usage

```
domain = lan.lxidomain
lan.lxidomain = domain
```

domain	The LXI domain number (0 to 255)

Details

This attribute sets the LXI domain number.

All outgoing LXI packets will be generated with this domain number. All inbound LXI packets will be ignored unless they have this domain number.

Example

`print(lan.lxidomain)`	Displays the LXI domain.

Also see

lan.restoredefaults() (on page 8-274)

lan.nagle

This attribute controls the state of the LAN Nagle algorithm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (lan.DISABLE)

Usage

```
state = lan.nagle
lan.nagle = state
```

state	`1` or `lan.ENABLE`: Enable the LAN Nagle algorithm for TCP connections `0` or `lan.DISABLE`: Disable the Nagle algorithm for TCP connections

Details

This attribute enables or disables the use of the LAN Nagle algorithm on transmission control protocol (TCP) connections.

Also see

lan.restoredefaults() (on page 8-274)

lan.reset()

This function resets the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.reset()
```

Details

This function resets the LAN interface. It performs the commands `lan.restoredefaults()` and `lan.applysettings()`.

Also see

lan.applysettings() (on page 8-267)
lan.restoredefaults() (on page 8-274)

lan.restoredefaults()

This function resets LAN settings to default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.restoredefaults()
```

Details

The settings that are restored are shown in the following table.

Settings that are restored to default	
Attribute	**Default setting**
`lan.config.dns.address[N]`	`"0.0.0.0"`
`lan.config.dns.domain`	`""`
`lan.config.dns.dynamic`	`lan.ENABLE`
`lan.config.dns.hostname`	`"K-<model number>-<serial number>"`
`lan.config.dns.verify`	`lan.ENABLE`
`lan.config.gateway`	`"0.0.0.0"`
`lan.config.ipaddress`	`"0.0.0.0"`
`lan.config.method`	`lan.AUTO`
`lan.config.subnetmask`	`"255.255.255.0"`
`lan.lxidomain`	`0`
`localnode.password`	`"admin"`

The `lan.restoredefaults()` function does not reset the LAN password. The `localnode.password` attribute controls the web password, which can be reset separately.

This command is run when `lan.reset()` is sent.

Example

`lan.restoredefaults()`	Restores the LAN defaults.

Also see

lan.reset() (on page 8-274)
localnode.password (on page 8-295)

lan.status.dns.address[N]

This attribute contains the DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
dnsAddress = lan.status.dns.address[N]
```

dnsAddress	DNS server IP address
N	Entry index (1, 2, or 3)

Details

This attribute is an array of DNS server addresses. The instrument can use up to three addresses.

Unused or disabled entries are returned as `"0.0.0.0"` when read. The *dnsAddress* returned is a string specifying the IP address of the DNS server in dotted decimal notation.

You can only specify two addresses manually. However, the instrument uses up to three DNS server addresses. If two are specified, only the one given by a DHCP server is used. If no entries are specified, up to three address given by a DHCP server are used.

The value of `lan.status.dns.address[1]` is referenced first for all DNS lookups. The values of `lan.status.dns.address[2]` and `lan.status.dns.address[3]` are referenced second and third, respectively.

Example

`print(lan.status.dns.address[1])`	Outputs DNS server address 1, for example: `164.109.48.173`

Also see

lan.status.dns.name (on page 8-276)

lan.status.dns.name

This attribute contains the present DNS fully qualified host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

hostName = `lan.status.dns.name`

hostName	Fully qualified DNS host name that can be used to connect to the instrument

Details

A fully qualified domain name (FQDN), sometimes referred to as an absolute domain name, is a domain name that specifies its exact location in the tree hierarchy of the Domain Name System (DNS).

A FQDN is the complete domain name for a specific computer or host on the LAN. The FQDN consists of two parts: the host name and the domain name.

If the DNS host name for an instrument is not found, this attribute stores the IP address in dotted decimal notation.

Example

`print(lan.status.dns.name)`	Outputs the dynamic DNS host name.

Also see

lan.config.dns.address[N] (on page 8-267)
lan.config.dns.hostname (on page 8-269)

lan.status.duplex

This attribute contains the duplex mode presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

duplex = `lan.status.duplex`

duplex	LAN duplex setting can be one of the following values: `0` or `lan.HALF`: half-duplex operation `1` or `lan.FULL`: full-duplex operation

Example

`print(lan.status.duplex)`	Outputs the present LAN duplex mode, such as: `1.00000e+00`

Also see

None

lan.status.gateway

This attribute contains the gateway address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

gatewayAddress = lan.status.gateway

gatewayAddress	LAN gateway address presently being used

Details

The value of *gatewayAddress* is a string that indicates the IP address of the gateway in dotted decimal notation.

Example

`print(lan.status.gateway)`	Outputs the gateway address, such as: `192.168.0.1`

Also see

lan.config.gateway (on page 8-271)

lan.status.ipaddress

This attribute contains the LAN IP address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

ipAddress = lan.status.ipaddress

ipAddress	LAN IP address specified in dotted decimal notation

Details

The IP address is a character string that represents the IP address assigned to the instrument.

Example

`print(lan.status.ipaddress)`	Outputs the LAN IP address currently in use, such as: `192.168.0.2`

Also see

lan.config.ipaddress (on page 8-271)

lan.status.macaddress

This attribute contains the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
macAddress = lan.status.macaddress
```

macAddress	The instrument MAC address

Details

The MAC address is a character string representing the instrument's MAC address in hexadecimal notation. The string includes colons that separate the address octets (see Example).

Example

`print(lan.status.macaddress)`	Outputs the MAC address of the instrument, for example: `00:60:1A:00:00:57`

Also see

None

lan.status.port.dst

This attribute contains the LAN dead socket termination port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.dst
```

port	Dead socket termination socket port number

Details

This attribute holds the TCP port number used to reset all other LAN socket connections.

To reset all LAN connections, open a connection to the DST port number.

Example

`print(lan.status.port.dst)`	Outputs the LAN dead socket termination port number, such as: `5.03000e+03`

Also see

None

lan.status.port.rawsocket

This attribute contains the LAN raw socket connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

port = lan.status.port.rawsocket

port	Raw socket port number

Details

Stores the TCP port number used to connect the instrument and to control the instrument over a raw socket communication interface.

Example

`print(lan.status.port.rawsocket)`	Outputs the LAN raw socket port number, such as: `5.02500e+03`

Also see

None

lan.status.port.telnet

This attribute contains the LAN Telnet connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

port = lan.status.port.telnet

port	Telnet port number

Details

This attribute holds the TCP port number used to connect to the instrument to control it over a Telnet interface.

Example

`print(lan.status.port.telnet)`	Get the LAN Telnet connection port number. Output: `2.30000e+01`

Also see

None

lan.status.port.vxi11

This attribute contains the LAN VXI-11 connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.vxi11
```

port	LAN VXI-11 port number

Details

This attribute stores the TCP port number used to connect to the instrument over a VXI-11 interface.

Example

`print(lan.status.port.vxi11)`	Outputs the VXI-11 number, such as: `1.02400e+03`

Also see

None

lan.status.speed

This attribute contains the LAN speed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
speed = lan.status.speed
```

speed	LAN speed in Mbps, either 10 or 100

Details

This attribute indicates the transmission speed currently in use by the LAN interface.

Example

`print(lan.status.speed)`	Outputs the instrument's transmission speed presently in use, such as: `1.00000e+02`

Also see

None

lan.status.subnetmask

This attribute contains the LAN subnet mask that is presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`mask = lan.status.subnetmask`

mask	A string specifying the subnet mask in dotted decimal notation

Details

Use this attribute to determine the present operating state of the LAN. This attribute will return the present LAN subnet mask value if the LAN is manually configured, or when DLLA or DHCP is used.

Example

`print(lan.status.subnetmask)`	Outputs the subnet mask of the instrument that is presently in use, such as: `255.255.255.0`

Also see

lan.config.subnetmask (on page 8-272)

lan.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`lan.trigger[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

`lan.trigger[5].assert()`	Creates a trigger with LAN packet 5.

Also see

lan.lxidomain (on page 8-273)
lan.trigger[N].clear() (on page 8-282)
lan.trigger[N].mode (on page 8-286)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].stimulus (on page 8-288)
lan.trigger[N].wait() (on page 8-291)
Understanding hardware value and pseudo line state

lan.trigger[N].clear()

This function clears the event detector for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].clear()
```

N	The LAN event number to clear (1 to 8)

Details

The trigger event detector enters the detected state when an event is detected. This function clears a trigger event detector and discards the previous history of the trigger packet.

This function clears all overruns associated with this LAN trigger.

Example

`lan.trigger[5].clear()`	Clears the event detector with LAN packet 5.

Also see

lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].stimulus (on page 8-288)
lan.trigger[N].wait() (on page 8-291)

lan.trigger[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

```
lan.trigger[N].connect()
```

N	The LAN event number (1 to 8)

Details

Prepares the event generator to send event messages. For TCP connections, this opens the TCP connection.

The event generator automatically disconnects when either the `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes for this event are changed.

Example

`lan.trigger[1].protocol = lan.MULTICAST` `lan.trigger[1].connect()` `lan.trigger[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

lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].ipaddress (on page 8-285)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].protocol (on page 8-287)
lan.trigger[N].stimulus (on page 8-288)
lan.trigger[N].wait() (on page 8-291)

lan.trigger[N].connected

This attribute stores the LAN event connection state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
connected = lan.trigger[N].connected
```

connected	The LAN event connection state: • `true`: Connected • `false`: Not connected
N	The LAN event number (1 to 8)

Details

This read-only attribute is set to `true` when the LAN trigger is connected and ready to send trigger events following a successful `lan.trigger[N].connect()` command; if the LAN trigger is not ready to send trigger events, this value is `false`.

This attribute is also `false` when either `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes are changed or the remote connection closes the connection.

Example

`lan.trigger[1].protocol = lan.MULTICAST` `print(lan.trigger[1].connected)`	Outputs `true` if connected, or `false` if not connected. Example output: `false`

Also see

lan.trigger[N].connect() (on page 8-283)
lan.trigger[N].ipaddress (on page 8-285)
lan.trigger[N].protocol (on page 8-287)

lan.trigger[N].disconnect()

This function disconnects the LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].disconnect()
```

N	The LAN event number (1 to 8)

Details

For TCP connections, this closes the TCP connection.

The LAN trigger automatically disconnects when either the `lan.trigger[N].protocol` or `lan.trigger[N].ipaddress` attributes for this event are changed.

Also see

lan.trigger[N].ipaddress (on page 8-285)
lan.trigger[N].protocol (on page 8-287)

lan.trigger[N].EVENT_ID

This constant is the event identifier used to route the LAN trigger to other subsystems (using stimulus properties).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
lan.trigger[N].EVENT_ID
```

N	The LAN event number (1 to 8)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to incoming LAN trigger packets.

Example

`digio.trigger[14].stimulus = lan.trigger[1].EVENT_ID`	Route occurrences of triggers on LAN trigger 1 to digital I/O trigger 14.

Also see

None

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Create configuration script Save setup	"0.0.0.0"

Usage

```
ipAddress = lan.trigger[N].ipaddress
lan.trigger[N].ipaddress = ipAddress
```

ipAddress	The LAN address for this attribute as a string in dotted decimal notation
N	A number specifying the LAN event number (1 to 8)

Details

Sets the IP address for outgoing trigger events.

Set to `"0.0.0.0"` for multicast.

After changing this setting, the `lan.trigger[N].connect()` command must be called before outgoing messages can be sent.

Example

`lan.trigger[3].protocol = lan.TCP` `lan.trigger[3].ipaddress = "192.168.1.100"` `lan.trigger[3].connect()`	Set the protocol for LAN trigger 3 to be `lan.TCP` when sending LAN triggers. Use IP address `"192.168.1.100"` to connect the LAN trigger.

Also see

lan.trigger[N].connect() (on page 8-283)

lan.trigger[N].mode

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	Instrument reset LAN trigger N reset Recall setup	Create configuration script Save setup	0 (lan.TRIG_EITHER)

Usage

```
mode = lan.trigger[N].mode
lan.trigger[N].mode = mode
```

mode	A number representing the trigger mode (0 to 7); see the **Details** section for more information
N	A number representing the LAN event number (1 to 8)

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger. These settings are intended to provide behavior similar to the digital I/O triggers.

LAN trigger mode values

Mode	Number	Trigger packets detected as input	LAN trigger packet generated for output with a…
`lan.TRIG_EITHER`	0	Rising or falling edge (positive or negative state)	negative state
`lan.TRIG_FALLING`	1	Falling edge (negative state)	negative state
`lan.TRIG_RISING`	2	Rising edge (positive state)	positive state
`lan.TRIG_RISINGA`	3	Rising edge (positive state)	positive state
`lan.TRIG_RISINGM`	4	Rising edge (positive state)	positive state
`lan.TRIG_SYNCHRONOUS`	5	Falling edge (negative state)	positive state
`lan.TRIG_SYNCHRONOUSA`	6	Falling edge (negative state)	positive state
`lan.TRIG_SYNCHRONOUSM`	7	Rising edge (positive state)	negative state

`lan.TRIG_RISING` and `lan.TRIG_RISINGA` are the same.

`lan.TRIG_RISING` and `lan.TRIG_RISINGM` are the same.

Use of either `lan.TRIG_SYNCHRONOUSA` or `lan.TRIG_SYNCHRONOUSM` over `lan.TRIG_SYNCHRONOUS` is preferred, as `lan.TRIG_SYNCHRONOUS` is provided for compatibility with other Keithley Instruments products.

Example

`print(lan.trigger[1].mode)`	Outputs the present LAN trigger mode of LAN event 1.

Also see

Digital I/O (on page 3-43)
TSP-Link system expansion interface (on page 7-45)

lan.trigger[N].overrun

This attribute contains the event detector's overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	LAN trigger N clear LAN trigger N reset Instrument reset Recall setup	Not applicable	Not applicable

Usage

```
overrun = lan.trigger[N].overrun
```

overrun	The trigger overrun state for the specified LAN packet (`true` or `false`)
N	A number representing the LAN event number (1 to 8)

Details

This attribute indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself. It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event.

It also is not an indication of an output trigger overrun.

Example

`overrun = lan.trigger[5].overrun` `print(overrun)`	Checks the overrun status of a trigger on LAN5 and outputs the value, such as: `false`

Also see

lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].clear() (on page 8-282)
lan.trigger[N].stimulus (on page 8-288)
lan.trigger[N].wait() (on page 8-291)

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Create configuration script Save setup	0 (lan.TCP)

Usage

```
protocol = lan.trigger[N].protocol
lan.trigger[N].protocol = protocol
```

protocol	The protocol to use for the trigger's messages: • `0` or `lan.TCP` • `1` or `lan.UDP` • `2` or `lan.MULTICAST`
N	A number representing the LAN event number (1 to 8)

Details

The LAN trigger listens for trigger messages on all supported protocols, but uses the designated protocol for sending outgoing messages. After changing this setting, `lan.trigger[N].connect()` must be called before outgoing event messages can be sent.

When the `lan.MULTICAST` protocol is selected, the `lan.trigger[N].ipaddress` attribute is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

`print(lan.trigger[1].protocol)`	Get LAN protocol to use for sending trigger messages for LAN event 1.

Also see

lan.trigger[N].connect() (on page 8-283)
lan.trigger[N].ipaddress (on page 8-285)

lan.trigger[N].pseudostate

This attribute sets the simulated line state for the LAN trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset LAN trigger N reset Recall setup	Create configuration script Save setup	1

Usage

```
pseudostate = lan.trigger[N].pseudostate
lan.trigger[N].pseudostate = pseudostate
```

pseudostate	The simulated line state (0 or 1)
N	A number representing the LAN event number (1 to 8)

Details

This attribute can be set to initialize the pseudo line state to a known value.

Setting this attribute does not cause the LAN trigger to generate any events or output packets.

Example

`print(lan.trigger[1].pseudostate)`	Get the present simulated line state for the LAN event 1.

Also see

None

lan.trigger[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	Instrument reset LAN trigger N reset Recall setup	Create configuration script Recall setup	0

Usage

```
triggerStimulus = lan.trigger[N].stimulus
lan.trigger[N].stimulus = triggerStimulus
```

triggerStimulus	The LAN event identifier used to trigger the event
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 *`triggerStimulus`* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID or 41 to 48`	The trigger event generated by the channel trigger *N*.
`digio.trigger[N].EVENT_ID or 1 to 14`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID or 39`	The trigger key (TRIG) on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH or 53`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW or 52`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH or 55`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW or 54`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID or 40`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID or 58 to 59`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID or 20 to 23`	A delay expired.
`tsplink.trigger[N].EVENT_ID or 17 to 19`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID or 29 to 36`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY or 24`	Scan ready event.
`scan.trigger.EVENT_SCAN_START or 25`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY or 28`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP or 56`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP or 50`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP or 26`	Scan complete event.
`scan.trigger.EVENT_IDLE or 27`	Idle event.
`schedule.alarm[N].EVENT_ID or 37 to 38`	Trigger event generated by the alarm *N*.

NOTE

Use one of the text trigger event IDs (for example, `digio.trigger[`*`N`*`].EVENT_ID`) to set the stimulus value rather than the numeric value. Doing this will make the code compatible for future upgrades.

Setting this attribute to zero disables automatic trigger generation.

If any events are detected prior to calling `lan.trigger[N].connect()`, the event is ignored and the action overrun is set.

Example

`lan.trigger[5].stimulus = trigger.timer[1].EVENT_ID`	Use timer 1 trigger event as the source for LAN packet 5 trigger stimulus.

Also see

lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].clear() (on page 8-282)
lan.trigger[N].connect() (on page 8-283)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].wait() (on page 8-291)

lan.trigger[N].wait()

This function waits for an input trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = lan.trigger[N].wait(timeout)
```

triggered	Trigger detection indication
N	The trigger packet over LAN to wait for (1 to 8)
timeout	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `lan.trigger[N].wait()` or `lan.trigger[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 = lan.trigger[5].wait(3)`	Wait for a trigger with LAN packet 5 with a timeout of 3 seconds.

Also see

lan.trigger[N].assert() (on page 8-281)
lan.trigger[N].clear() (on page 8-282)
lan.trigger[N].overrun (on page 8-287)
lan.trigger[N].stimulus (on page 8-288)

localnode.define.*

These constants indicate the number of available features (of each feature type) for each local node instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
CONSTANT (R)	- -			
.MAX_TIMERS	Yes			
.MAX_DIO_LINES	Yes			
.MAX_TSPLINK_TRIGS	Yes			
.MAX_BLENDERS	Yes			
.MAX_BLENDER_INPUTS	Yes			
.MAX_LAN_TRIGS	Yes			

Usage

```
maxNumber = localnode.define.MAX_TIMERS
maxNumber = localnode.define.MAX_DIO_LINES
maxNumber = localnode.define.MAX_TSPLINK_TRIGS
maxNumber = localnode.define.MAX_BLENDERS
maxNumber = localnode.define.MAX_BLENDER_INPUTS
maxNumber = localnode.define.MAX_LAN_TRIGS
maxNumber = localnode.define.MAX_CHANNEL_TRIGS
```

maxNumber	A variable assigned the value of the constant.; the constant equals the local node instrument's maximum number available for the specified feature

Details

These read-only constants indicate the following types of features: timers, digital input/output lines, triggers, and blenders. They provide the number of features available (which is the maximum) for the specified local node feature.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].define.MAX_TIMERS`.

Example 1

`maxNumber = localnode.define.MAX_TIMERS`	Reads the maximum number of timers that are available for the presently active instrument.

Also see

None

localnode.description

This attribute stores a user-defined description and mDNS service name of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	Instrument specific (see **Details**)

Usage

```
localnode.description = description
description = localnode.description
```

description	User-defined description and mDNS service name of the instrument; use a string of 63 characters or less

Details

This attribute stores a string that contains a description of the instrument. This value appears on instrument's LXI welcome page. The value of this attribute is also used as the instrument's mDNS service name.

This attribute's default value contains `Keithley` *`ModelNumber`* *#SSSSSSSS*, where: *`ModelNumber`* is the instrument's model number, and *#SSSSSSSS* is the instrument's eight-digit serial number. You can change it to a value that makes sense for your system. Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero, or one consisting entirely of whitespace characters) will revert the description to the factory default value.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].description`.

Example

`description = "System in Lab 05"` `localnode.description = description`	Set `description` to "System in Lab 05".

Also see

None

localnode.emulation

This attribute sets the instrument to report the model number as 3706 instead of 3706A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Nonvolatile memory	localnode.OFF

Usage

```
value = localnode.emulation
localnode.emulation = value
```

value	`0` or `localnode.OFF`: No emulation (model number is reported as 3706A). `1` or `localnode.EMULATION_3706`: Reports the model number as 3706.

Details

This command needs to be set if you replace a Model 3706 with a Model 3706A in a system where computer drivers may be querying the model. This can occur if you replace a Model 3706 with a Model 3706A in an existing system, or if you duplicate a system but use a Model 3706A instead of a Model 3706.

When this command is set to `localnode.EMULATION_3706`, the model number is reported as a 3706 when you send a request with a command such as `localnode.model` or `*idn?`. This allows drivers that query the model number to continue to operate normally.

NOTE

All other Model 3706A behavior is the same. Emulation mode does not affect the changes to the IEEE-1588 features or the response times that occurred with the update from the Model 3706 to the Model 3706A.

This setting is preserved through a power cycle and instrument reset.

Example

`localnode.emulation = localnode.EMULATION_3706`	Sets the Model 3706A for Model 3706 emulation.

Also see

localnode.model (on page 8-295)

localnode.linefreq

This attribute contains the power line frequency setting used for NPLC calculations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	60

Usage

```
frequency = localnode.linefreq
```

frequency	An integer representing the instrument's detected line frequency

Details

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].linefreq`.

Example

`frequency = localnode.linefreq`	Reads line frequency setting.

Also see

None

localnode.model

This attribute stores the model number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`model = localnode.model`

model	The model number of the instrument

Details

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].model`.

Example

`print(localnode.model)`	Outputs the model number of the local node. For example:

Also see

localnode.serialno (on page 8-299)

localnode.password

This attribute stores the remote access password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	LAN reset	Nonvolatile memory	"admin"

Usage

`localnode.password = "password"`

password	A string that contains the remote interface password

Details

This write-only attribute stores the password that is set for any remote interface. When password usage is enabled (`localnode.passwordmode`), you must supply a password to change the configuration or to control an instrument from a web page or other remote command interface.

The instrument continues to use the old password for all interactions until the command to change it executes. When changing the password, give the instrument time to execute the command before attempting to use the new password.

You can retrieve the password from the front panel through **MENU > LAN > STATUS > PASSWORD**.

The password can be reset by resetting the LAN from the front panel or by using the `lan.reset()` command.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].password`.

Example

`localnode.password = "N3wpa55w0rd"`	Changes the remote interface password to `N3wpa55w0rd`.

Also see

lan.reset() (on page 8-274)
localnode.passwordmode (on page 8-296)

localnode.passwordmode

This attribute stores the remote access password enable mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	1 (localnode.PASSWORD_WEB)

Usage

```
mode = localnode.passwordmode
localnode.passwordmode = mode
```

mode	The remote password enable mode

Details

This attribute controls if and where remote access passwords are required. Set this attribute to one of the values below to enable password checking:

`localnode.PASSWORD_NONE` or `0`: Disable passwords everywhere

`localnode.PASSWORD_WEB` or `1`: Use passwords on the web interface only

`localnode.PASSWORD_LAN` or `2`: Use passwords on the web interface and all LAN interfaces

`localnode.PASSWORD_ALL` or `3`: Use passwords on the web interface and all remote command interfaces

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].passwordmode`.

Example

`mode = localnode.PASSWORD_WEB` `localnode.passwordmode = mode`	Sets value of `mode` to `PASSWORD_WEB`. Allows use of passwords on the web interface only.

Also see

localnode.password (on page 8-295)

localnode.prompts

This attribute sets and reads the local node prompting state (enabled or disabled).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Prompting state (`0` to disable or `1` to enable)

Details

The command messages do not generate prompts. The instrument generates prompts in response to command messages.

When the prompting mode is enabled (set to 1), the instrument generates prompts in response to command messages. There are three prompts that might be generated:

- **TSP>** is the standard prompt. This prompt indicates that everything is normal and the command is done processing.
- **TSP?** is issued if there are entries in the error queue when the prompt is issued. Like the TSP> prompt, it indicates the command is done processing. It does not mean the previous command generated an error, only that there are still errors in the queue when the command was done processing.
- **>>>>** is the continuation prompt. This prompt is used when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the current command.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].prompts`.

NOTE

Do not disable prompting when using Test Script Builder. Test Script Builder requires prompts and sets the prompting mode behind the scenes. If you disable prompting, using Test Script Builder causes the instrument to stop responding because it is waiting for the prompt that lets it know that the command is done executing.

Example

`localnode.prompts = 1`	Enable prompting.

Also see

localnode.prompts4882 (on page 8-297)
localnode.showerrors (on page 8-300)
tsplink.reset() <CTS2600B_only_start>tsplink.reset() <CTS2600B_only_end>

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)	Yes	Power cycle	Not saved	1 (enabled)

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting	IEEE Std 488.2 prompting mode

Details

When set to 1, the IEEE Std 488.2 common commands generate prompts if prompting is enabled with the `localnode.prompts` attribute. If set to 1, limit the number of `*trg` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When set to 0, IEEE Std 488.2 common commands will not generate prompts. When using the `*trg` command with a script that executes `trigger.wait()` repeatedly, set `localnode.prompts4882` to 0 to avoid problems associated with the command interface input queue filling.

This attribute resets to the default value each time the instrument power is cycled.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].prompts4882`.

Example

`localnode.prompts4882 = 0`	Disables IEEE Std 488.2 common command prompting.

Also see

localnode.prompts (on page 8-296)

localnode.reset()

This function resets the local node instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
localnode.reset()
```

Details

If you want to reset a specific instrument or a subordinate node, use the `node[X].reset()` command.

A local node reset includes a `channel.reset("allslots")`, `dmm.reset("all")` and a `scan.reset()`. In addition:

- Other settings are restored back to factory default settings
- Existing channel patterns and DMM configurations are deleted
- All channels and backplane relays are opened
- The dmm function is `"dcvolts"`
- User-created reading buffers are deleted

A `localnode.reset()` is different than a `reset()` because `reset()` resets the entire system.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example `node[5].reset()`.

Example

`localnode.reset()`	Resets the local node.

Also see

channel.reset() (on page 8-87)
dmm.reset() (on page 8-228)
reset() (on page 8-317)
scan.reset() (on page 8-334)

localnode.revision

This attribute stores the firmware revision level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

revision = localnode.revision

revision	Firmware revision level

Details

This attribute indicates the revision number of the firmware that is presently running in the instrument.

When using this command from a remote node, `localnode` should be replaced with the node reference. For example, `node[5].revision`.

Example

`print(localnode.revision)`	Outputs the present revision level. Sample output: `01.50b`

Also see

localnode.description (on page 8-293)
localnode.model (on page 8-295)
localnode.serialno (on page 8-299)

localnode.serialno

This attribute stores the instrument's serial number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

serialno = localnode.serialno

serialno	The serial number of the instrument

Details

This indicates the instrument serial number.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].serialno`.

Example

`display.clear()` `display.settext(localnode.serialno)`	Clears the instrument's display. Places the instrument's serial number on the top line of its display.

Also see

localnode.description (on page 8-293)
localnode.model (on page 8-295)
localnode.revision (on page 8-299)

localnode.showerrors

This attribute sets whether or not the instrument automatically sends generated errors.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	0 (disabled)

Usage

```
errorMode = localnode.showerrors
localnode.showerrors = errorMode
```

errorMode	Enables (1) or disables (0) the show errors state

Details

If this attribute is set to `1`, the instrument automatically sends any generated errors stored in the error queue, and then clears the queue. Errors are processed after executing a command message (just before issuing a prompt, if prompts are enabled).

If this attribute is set to `0`, errors are left in the error queue and must be explicitly read or cleared.

When using this command from a remote node, `localnode` should be replaced with the node reference, for example, `node[5].showerrors`.

Example

`localnode.showerrors = 1`	Enables sending of generated errors.

Also see

localnode.prompts (on page 8-296)

makegetter()

This function creates a function to get the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
getter = makegetter(table, attributeName)
```

getter	The return value
table	Read-only table where the attribute is located
attributeName	A string representing the name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the function created with `makegetter()` executes faster than accessing the attribute directly.

Creating a getter function is only useful if it is going to be called several times. Otherwise, the overhead of creating the getter function outweighs the overhead of accessing the attribute directly.

Example

`getrange = makegetter(dmm, "range")` `-- (intervening code)` `r = getrange()`	Create a getter function called `getrange`. When `getrange()` is called, it returns the value of `dmm.range` and assigns it to the variable `r`.

Also see

makesetter() (on page 8-301)

makesetter()

This function creates a function that, when called, sets the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`setter` `= makesetter(`*`table`*`,` *`attributeName`*`)`

`setter`	Function that sets the value of the attribute
`table`	Read-only table where the attribute is located
`attributeName`	The string name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the *`setter`* function will execute faster than accessing the attribute directly.

Creating a *`setter`* function is only useful if it is going to be called several times. If you are not calling the *`setter`* function several times, it is more efficient to access the attribute directly.

Example

`setrange = makesetter(dmm, "range")` `setrange(5)`	Use `setrange` with a value of 5 to set `dmm.range` for the currently selected function.

Also see

makegetter() (on page 8-300)

memory.available()

This function reads and returns the amount of memory that is available in the instrument overall for storing user scripts and channel patterns and for user-defined DMM configurations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

memoryAvailable = `memory.available()`

memoryAvailable	Comma-delimited string with percentages for available memory; the format is `systemMemory`, `scriptMemory`, `patternMemory`, `configurationMemory`, where: • `systemMemory`: The percentage of memory available in the instrument • `scriptMemory`: The percentage of memory available in the instrument to store user scripts • `patternMemory`: The percentage of memory available in the instrument to store channel patterns • `configurationMemory`: The percentage of memory available to store DMM configurations

Details

Use this function to view the available memory in the overall instrument as well as the memory available for storing user scripts, channel patterns, and user DMM configurations.

The response to this function is a single string that returns the overall instrument memory available, script memory available,channel pattern memory available, and DMM configuration memory available as comma-delimited percentages.

Example: Available memory

`memoryAvailable = memory.available()` `print(memoryAvailable)`	Reads and returns the amount of memory available in the instrument. Output: `51.56, 92.84, 100.00, 100.00` You can also use: `print(memory.available())`

Example: After recalling a setup

`setup.recall(1)` `print(memory.available())`	Reads and returns the amount of memory available in the instrument after a setup is recalled. Output: `11.13, 92.84, 0.16, 97.03`

Example: Used and available memory

`print("Memory used:", memory.used())` `print("Memory available: ",` `   memory.available())`	Reads and returns the amount memory used and memory available percentages. Output: `Memory used: 69.14, 0.16, 12.74,  1.04` `Memory available: 30.86, 99.84, 87.26,` `   98.96`

Also see

memory.used() (on page 8-303)

memory.used()

This function reports the amount of memory used in the instrument overall and for user scripts, storing channel patterns, and storing user DMM configurations.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
memoryUsed = memory.used()
```

memoryUsed	A comma-delimited string with percentages for used memory; the format is `systemMemory`, `scriptMemory`, `patternMemory`, `configurationMemory`, where: • `systemMemory`: The percentage of memory used in the instrument • `scriptMemory`: The percentage of memory used in the instrument to store user scripts • `patternMemory`: The percentage of memory used in the instrument to store channel patterns • `configurationMemory`: The percentage of memory used to store DMM configurations

Details

Use this function to view the used memory in the overall instrument, as well as the memory used for storing user scripts, channel patterns, and user DMM configurations.

The response to this function is a single string that shows the overall instrument memory used, as well as the script memory used, channel pattern memory used, and DMM configuration memory used as comma-delimited percentages.

Example

`MemUsed = memory.used()` `print(MemUsed)`	Reads the memory used in the instrument and returns out the percentages. Output: `69.14, 0.16, 12.74, 1.04`

Also see

memory.available() (on page 8-302)

node[N].execute()

This function starts test scripts on a remote TSP-Link node. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

`node[N].execute(scriptCode)`

N	The node number of this instrument (1 to 64)
scriptCode	A string containing the source code

Details

This command is only applicable to TSP-Link systems. You can use this command to use the remote master node to run a script on the specified node. This function does not run test scripts on the master node; only on the subordinate node when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

Example 1

`node[2].execute(sourcecode)`	Runs script code on node 2. The code is in a string variable called `sourcecode`.

Example 2

`node[3].execute("x = 5")`	Runs script code in string constant (`"x = 5"`) to set x equal to 5 on node 3.

Example 3

`node[32].execute(TestDut.source)`	Runs the test script stored in the variable `TestDut` (previously stored on the master node) on node 32.

Also see

Introduction to TSP advanced features
tsplink.group (on page 8-436)

node[N].getglobal()

This function returns the value of a global variable. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`value = node[N].getglobal(name)`

value	The value of the variable
N	The node number of this instrument (1 to 64)
name	The global variable name

Details

This function retrieves the value of a global variable from the run-time environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link® network.

Example

`print(node[5].getglobal("test_val"))`	Retrieves and outputs the value of the global variable named test_val from node 5.

Also see

node[N].setglobal() (on page 8-305)
Introduction to TSP advanced features

node[N].setglobal()

This function sets the value of a global variable. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
node[N].setglobal(name, value)
```

N	The node number of this instrument (1 to 64)
name	The global variable name to set
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link®.

Example

`node[3].setglobal("x", 5)`	Sets the global variable x on node 3 to the value of 5.

Also see

node[N].getglobal() (on page 8-304)
Using TSP to run test scripts simultaneously

opc()

This function sets the operation complete status bit when all overlapped commands are completed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
opc()
```

Details

This function causes the operation complete bit in the Standard Event Status Register to be set when all previously started local overlapped commands are complete.

Note that each node 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.

Also see

Status model (on page 6-18, on page D-1, "*Status Byte Register overview*" on page D-4)
waitcomplete() (on page 8-466)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments are separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want use Lua formatting, print the return value from the `tostring()` function.

Example 1

`x = 10` `print(x)`	Example of an output response message: `1.00000e+01` Note that your output might be different if you set your ASCII precision setting to a different value.

Example 2

`x = 10` `print(tostring(x))`	Example of an output response message: `10`

Also see

format.asciiprecision (on page 8-255)

printbuffer()

This function prints data from tables or reading buffer subtables.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

startIndex	Beginning index of the buffer to print
endIndex	Ending index of the buffer to print
bufferVar	Name of first table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVar2	Second table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
bufferVarN	The last table or reading buffer subtable to print; may be a default buffer (`defbuffer1` or `defbuffer2`) or a user-defined buffer
...	One or more tables or reading buffer subtables separated with commas

Details

The correct usage of this function for a buffer containing *n* elements is:

1 ≤ *startIndex* ≤ *endIndex* ≤ *n*

Where *n* refers to the index of the last entry in the tables to be printed.

If *endIndex* < *startIndex* or *n* < *startIndex*, no data is printed. If *startIndex* ≤ 1, 1 is used as *startIndex*. If *n* < *endIndex*, *n* is used as *endIndex*.

When any given reading buffers are used in overlapped commands that have not yet completed (at least to the desired index), this function outputs data as it becomes available.

When there are outstanding overlapped commands to acquire data, *n* refers to the index that the last entry in the table will have after all the readings have completed.

If you pass a reading buffer instead of a reading buffer subtable, the default subtable for that reading buffer will be used.

This command generates a single response message that contains all data. The response message is stored in the output queue.

The `format.data` attribute controls the format of the response message.

The following bufferVar attributes can be used with the print buffer command.

bufferVar.dates, bufferVar.fillmode, bufferVar.formattedreadings, bufferVar.fractionalseonds, bufferVar.logstate, bufferVar.seconds, buffer.units, bufferVar.relativetimestamps, bufferVar.seconds, bufferVar.sourcestatuses, bufferVar, sourceunits, bufferVar.sourcevalues, bufferVar.statuses, bufferVar.times, bufferVar.timestamps, bufferVar.units

If you are using the printbuffer() command to print multiple readings from multiple buffers, the readings must be paired: buffer 1 readings, buffer 2 readings, next buffer 1 readings, next buffer 2 readings and so on. For example, when you are storing voltage readings in one buffer and current readings in another buffer, keep them paired in the response. Or, when you are just storing different readings in multiple buffer keep then paired in response.

Example 1

```
reset()
testData = buffer.make(200)
smu.measure.count = 6
smu.measure.read(testData)
smu.measure.read(defbuffer1)
format.data = format.ASCII
format.asciiprecision = 6
printbuffer(1, testData.n, testData.readings, testData.units,
   testData.relativetimestamps)
for x = 1, testData.n do
printbuffer(x,x,testData, testData.units, testData.relativetimestamps)
end
```

This assumes that `testData` is a valid reading buffer in the run-time environment. The use of `testData.n` (*`bufferVar`*`.n`) indicates that the instrument should output all readings in the reading buffer. In this example, `testBuffer.n` equals 6.

```
Example of output data
1.37000e-11, 4.84914e-11, 4.84921e-11, 4.84888e-11, 4.84859e-11, 2.76175e-11
```

Example 2

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

This example output data includes readings, units and relative timestamps.

```
1.37000e-11, Current, 0.00000e+00, 4.84914e-11, Current, 5.28490e-02, 4.84921e-
   11, Current, 1.05692e-01, 4.84888e-11, Current, 1.58533e-01, 4.84859e-11,
   Current, 2.11381e-01, 2.76175e-11, Current, 2.64231e-01
```

Example 3

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

Example output data showing readings, units and relative timestamps in printed in rows.

```
1.37000e-11, Current, 0.00000e+00
4.84914e-11, Current, 5.28490e-02
4.84921e-11, Current, 1.05692e-01
4.84888e-11, Current, 1.58533e-01
4.84859e-11, Current, 2.11381e-01
2.76175e-11, Current, 2.64231e-01
```

Also see

bufferVar.n
bufferVar.readings
format.asciiprecision
format.byteorder
format.data
printnumber() (on page 8-310)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

`format.asciiprecision = 10` `x = 2.54` `printnumber(x)` `format.asciiprecision = 3` `printnumber(x, 2.54321, 3.1)`	Configure the ASCII precision to 10 and set x to 2.54. Read the value of x based on these settings. Change the ASCII precision to 3. View how the change affects the output of x and some numbers. Output: `2.540000000e+00` `2.54e+00, 2.54e+00, 3.10e+00`

Also see

format.asciiprecision (on page 8-255)
format.byteorder (on page 8-256)
format.data (on page 8-257)
print() (on page 8-306)
printbuffer() (on page 8-307)

ptp.domain

This attribute describes the IEEE Std 1588-2008 precision time protocol (PTP) domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Never	Nonvolatile memory	0

Usage

```
value = ptp.domain
ptp.domain = value
```

value	0 = default domain 1 = alternate domain 1 2 = alternate domain 2 3 = alternate domain 3 4 – 127 = user-defined domains 128 – 255 = Reserved

Details

Only instruments in the same domain will interact with each other in the IEEE-1588 PTP.

Example

`ptp.domain=1` `print(ptp.domain)`	Sets the ptp domain to 1 (alternate domain 1) and prints the result. Output: `1`

Also see

Not applicable

ptp.ds.info()

This function is a read-only string that returns the settings of the different data sets (DS) associated with the IEEE-1588 2008 specification.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
ptp.ds.info()
```

Details

The following data sets are returned:

- Current
- Default
- Parent
- Time properties
- Port
- Foreign master

For more detailed information regarding field information, refer to the IEEE-1588 2008 specification.

Example

`print(ptp.ds.info())`	Output:

```
Current DS
         Steps removed:  0
    Offset from Master:  0.000000000
       Mean Path Delay:  0.000000000

Default DS
       Number of Ports:  1
        Two Step Clock:  T
            Priority 1:  128
            Priority 2:  128
                Domain:  0
        Clock Identity:  12 34 56 FF FE 65 43 21
    Clock Qual - Class:  248
 Clock Qual - Accuracy:  254
 Clock Qual - Variance:  0
            Slave Only:  F

Parent DS
           Parent Stats:  F
  Parent Clock Identify:  12 34 56 FF FE 65 43 21
   Parent Port Identify:  0
      Parent Offset Var:  65535
Parent Phase Chnge Rate:  2147483647
          GM Priority 1:  128
          GM Priority 2:  128
   GM Clck Qual - Class:  248
GM Clck Qual - Accuracy:  254
GM Clck Qual - Variance:  0
      GM Clock Identify:  12 34 56 FF FE 65 43 21

Time Properties DS
     Current UTC Offset:  0
                Leap 59:  F
                Leap 61:  F
Current UTC Offset Vald:  T
          PTP Timescale:  T
         Time Traceable:  F
    Frequency Traceable:  F
            Time Source:  Internal Oscillator

Port DS
         Clock Identify:  12 34 56 FF FE 65 43 21
          Port Identify:  1
             Port State:  6
 Log Mn Delay Req Intrvl: 4
   Peer mean Path Delay:  0
  Log Announce Interval:  1
 Announc Receipt Timeout: 3
      Log Sync Interval:  0
        Delay Mechanism:  E2E
Log Mn PDelay Rq Intrvl:  0
         Version Number:  2

Foreign Master DS 1
      Announce Messages:  2
 Frgn Mstr Clock Idntfy:  00 60 1A FF FE 01 54 29
  Frgn Mstr Port Idntfy:  1
```

Also see

Not applicable

ptp.enable

This attribute enables or disables the precision time protocol (PTP) described in IEEE-1588 on the Series 3700A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0 (ptp.OFF)

Usage

```
state = ptp.enable
ptp.enable = state
```

state	Disable the ptp protocol: `ptp.OFF` or `0` Enable the ptp protocol: `ptp.ON` or `1`

Details

From the factory, this attribute is disabled (`ptp.OFF`). After setting this attribute, it is saved in nonvolatile memory, and that setting value is recalled the next time the instrument is powered on.

Example

`ptp.enable=1` `print(ptp.enable)`	Output: `1.000000000e+00`

Also see

Not applicable

ptp.portstate

This attribute is a read-only value that indicates the state of the IEEE-1588 precision time protocol (PTP).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Never	Not applicable	Not applicable

Usage

```
state = ptp.portstate
```

state	ptp.INITIALIZING (0) ptp.FAULTY (1) ptp.DISABLE (2) ptp.LISTENING (3) ptp.PRE_MASTER (4) ptp.MASTER (5) ptp.PASSIVE (6) ptp.UNCALIBRATED (7) ptp.SLAVE (8) ptp.UNKNOWN (9)

`print(ptp.portstate)`	Output (this output indicates that PTP is disabled): `2.000000000e+00`

Also see

ptp.enable (on page 8-314)

ptp.slavepreferred

This attribute describes whether you prefer to have the instrument be a subordinate clock or not.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Never	Nonvolatile memory	false (disabled)

Usage

```
value = ptp.slavepreferred
ptp.slavepreferred = value
```

value	`true`: Disabled. `false`: Enabled.

Details

From the factory, this attribute is false. After you set this attribute, it is saved in nonvolatile memory. That setting is recalled the next time the instrument is powered up.

Example

`ptp.slavepreferred = TRUE` `print(ptp.slavepreferred)`	Set the instrument to be a subordinate clock. Check to see if the instrument is a subordinate clock. Output: `true`

Also see

Not applicable

ptp.time()

This function is a read-only string that returns the PTP time in seconds and fractionalseconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
ptp.time()
```

Example

`sec,fraction=ptp.time()` `print(sec+fraction)`	Output: `1.306440045e+09`

Also see

Not applicable

ptp.utcoffset

This attribute describes the offset, in seconds, between UTC and PTP.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	See Details	Nonvolatile memory	See **Details**

Usage

```
value = ptp.utcoffset
ptp.utcoffset = value
```

value	The offset in seconds

Details

If the instrument is a subordinate, the `ptp.utcoffset` value is from the master. If the value is from the master, the setting is overwritten on the next synchronization. The Series 3700A does not keep track of this value through a power cycle (that is, it defaults to 0 if the 3700 is the master). The `ptp.utcoffset` is only non-zero if the Series 3700A communicates to a master clock that is aware of the difference between PTP and UTC time.

You can only write to this command if the Series 3700A is the master. If the Series 3700A is not the master , an error is generated when you try to write to the Series 3700A.

The Series 3700A is not time-zone aware, so UTC time is presented as the local time.

UTC Time = PTP Time – UTC Offset

Example

`ptp.utcoffset=33` `print(ptp.utcoffset)`	Sets the UTC offset to 33 seconds. Output: `3.300000000e+01`

Also see

Not applicable

reset()

This function resets commands to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
reset()
reset(system)
```

system	`true`: If the node is the master, the entire system is reset `false`: Only the local group is reset

Details

The `reset()` command in its simplest form resets the entire TSP-enabled system, including the controlling node and all subordinate nodes.

If you want to reset a specific instrument, use either the `localnode.reset()` or `node[X].reset()` command. Use the `localnode.reset()` command for the local instrument. Use the `node[X].reset()` command to reset an instrument on a subordinate node.

When no value is specified for *`system`*, the default value is `true`.

You can only reset the entire system using `reset(true)` if the node is the master. If the node is not the master node, executing this command generates an error.

Example

`reset(true)`	If the node is the master node, the entire system is reset; if the node is not the master node, an error is generated.

Also see

localnode.reset() (on page 8-298)

scan.abort()

This function aborts a running background scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.abort()
```

Details

If no scan is running, the call to this function is ignored.

NOTE

When a scan is aborted, the channels remain in the opened or closed states that they were in when the scan was aborted.

Example

`scan.background()` `scan.abort()`	Starts background scan, and then aborts the scan.

Also see

scan.background() (on page 8-323)
Scanning and triggering (on page 3-1)

scan.add()

This function adds a scan step to the scan list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.add(channelList)
scan.add(<channelList>, <dmmConfig>)
scan.add(<channelList>, <width>)
```

channelList	String specifying channels to add using normal channel list syntax
dmmConfig	String listing the DMM configuration to use with items in *channelList*
width	Value that specifies the width of the channel read to use with items in *channelList*

Details

Use this function to add channels and channel patterns to the present scan list. If the scan list does not exist, it also creates a scan list. See `scan.create()` for information about creating a scan list.

Channels and channel patterns added using the `scan.add()` function are added to the end of the present list (appended) in the same order as specified in *channelList*. In addition, the added channels are scanned in the order specified in *channelList*. Specifying multiple channels in *channelList* adds multiple steps to the scan.

Each channel's or channel pattern's configuration, associated with `dmm.setconfig()` and `dmm.getconfig()`, is used unless the optional *dmmConfig* parameter is specified. Specifying the *dmmConfig* parameter temporarily overrides the channel's (or channel pattern's) associated configuration. Specifying *dmmConfig* does not modify the assigned configuration of a channel or channel pattern.

The scan list of channels (or channel patterns) is not updated if an error occurs during processing of the function. However, because each channel is added as a separate step when you add multiple channels to *channelList*, steps that were already added to the scan list update, even if an error is detected.

For digital I/O or totalizer channels, each created scan step instructs the scan to read the selected channel and then save the value into the specified reading buffer. If you do not specify a reading buffer, the channel is read but the value is not saved.

The *width* parameter is valid for digital I/O type channels. Widths of 1, 2, 3, or 4 are supported. If specified, the scan can read up to four consecutive channels simultaneously, and then saves the resulting value into the specified reading buffer.

DAC channels are not supported.

Measurement time stamps may vary from channel read time stamps because of the way different channel types generate reading buffer time stamps.

Example 1

`scan.create("3001:3010", "DCV")`	For this example, assume `"DCV"` is a previously defined user configuration for DC volts. Clears the old scan list and creates a new scan list with each channel (1 to 10 on slot 3) using DCV as the overriding DMM configuration.

Example 2

`scan.add("3001:3010", "2wire")`	For this example, assume `"2wire" is a` previously defined user configuration for 2-wire ohms. Uses `2wire` for all 10 channels and adds them to the end of the existing scan list.

Example 3

`scan.create("")`	Clears the old scan list and creates a new empty scan list.

Example 4

```
for chan = 3001, 3010 do

   scan.add("" .. chan, "DCV")

   scan.add("" .. chan, "2wire")

end
```

For this example, assume `"DCV"` is a previously defined user DC volts configuration, and `"2wire"` is a previously defined user 2-wire ohms configuration.

Adds channels 3001 through 3010 to the end of the existing scan list. This loops through the channels twice, adding channels to the scan list twice. The first time, it adds `"DCV"` for a channel. The second time, it adds `"2wire"` for that channel. The first parameter `("" .. chan)` converts the chan number to a string.

Also see

scan.create() (on page 8-325)
dmm.getconfig() (on page 8-189)
dmm.setconfig() (on page 8-237)
Reading buffers (on page 3-55, on page 3-49)
Scanning and triggering (on page 3-1)

scan.addimagestep()

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			

Usage

```
scan.addimagestep(channelList)
scan.addimagestep(channelList, dmmConfig)
```

channelList	String specifying a list of channels
dmmConfig	String specifying a DMM configuration

Details

This function adds a list of channels to be closed simultaneously in a single step of a scan. An optional DMM configuration can be added to force the scan to take a measurement during the same step.

This function is an advanced command; the *channelList* parameter must specify appropriate relays to support the requested DMM configuration, or an invalid measurement will result.

Unlike `scan.add()`, the paired channels and backplanes necessary for measurement are not automatically added to the step. Use the `channel.setpole()` command to indicate if the paired channel should be added or not. Backplanes assigned to channels by the `channel.setbackplane()` command are not added to the image step automatically. For example, if a measurement is taken on a 4-wire ohms configuration without designating 4-pole with the `channel.setpole()` command, the corresponding paired channels and backplanes will not be added, and the specified *dmmConfig* will not cause additional relay closures as it normally would.

If you have changed the pole setting on any of the channels in the list (using `channel.setpole()`), an additional paired channel is added or removed, as appropriate. For example, to ensure that the proper channels close to enable a 4-wire measurement, set the pole setting in addition to using the 4-wire ohms DMM configuration.

When a DMM configuration (other than `"nofunction"`) is specified, the instrument will take the appropriate measurement based on the function set in the configuration; if no DMM configuration is specified with the command, no measurement will be taken.

Example

`scan.addimagestep("1001", "dcvolts")`	Adds a single step that closes Channel 1001 and takes a DC voltage measurement. Note that the voltage measurement will be inaccurate if this is the only step in the scan (because the backplane channels are not closed).
`scan.addimagestep("1001, 1911", "dcvolts")`	Adds a single step that closes Channels 1001 and 1911, and then takes a DC voltage measurement.
`channel.setpole("1001", 4)` `scan.addimagestep("1001, 1911", "dcvolts")`	Set Channel 1001 to 4-pole operation. Adds a single step that closes Channels 1001, 1031, and 1911, and then takes a DC voltage measurement.

`scan.addimagestep("1101, 2202, 1911", "dcvolts")` `scan.addimagestep("1102, 2202, 1911", "dcvolts")` `scan.addimagestep ("1103, 2202, 1911","dcvolts")`	Adds three steps with the following actions: • Closes Channels 1101, 2202, and 1911, and then takes a DC voltage measurement. • Opens Channel 1101, closes Channel 1102 and maintains Channel 1911 and 2202 closed, and then takes a DC voltage measurement. • Opens Channel 1102, closes Channel 1103 and maintains Channel 1911 and 2202 closed, and then takes a DC voltage measurement.

Also see

channel.setbackplane() (on page 8-90)
channel.setpole() (on page 8-101)
scan.add() (on page 8-319)
Scanning and triggering (on page 3-1)

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)
scan.addwrite(channelList, writeValue, width)
```

channelList	String specifying channels to add using normal channel list syntax
writeValue	The value to write to the channel for this scan step
width	Specifies the width of the channel write to use with items in *channelList*

Details

This function is similar to issuing `channel.write()` at the scan step. Specifying multiple channels in *channelList* causes multiple steps to be added to the scan.

For digital I/O channels, only a width of 1, 2, 3, or 4 is supported. Any information (bits) greater than the specified width are ignored. Values written to inputs are ignored. If no specified channel is set for output, an error is generated. If a width crosses channels, only the channels set to output are affected.

For backplane and switch channels, there is no valid behavior. Calling on a specific channel generates an error.

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

`scan.addwrite("6001, 6003, 6005", 21845, 2)`	Assume a Model 3750 in slot 6. Add to existing scan list channels 1, 3 and 5 on slot 6 to write a 16-bit hex value of hexadecimal 5555 (decimal 21845).

Also see

Scanning and triggering (on page 3-1)

scan.background()

This function starts a scan and runs the scan in the background.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
state, scanCount, stepCount, reading = scan.background()
state, scanCount, stepCount, reading = scan.background(bufferVar)
```

state, scanCount, stepCount, reading = scan.background(bufferVar)

state	The result of scanning: `scan.EMPTY` or `0` `scan.BUILDING` or `1` `scan.RUNNING` or `2` `scan.ABORTED` or `3` `scan.FAILED` or `4` `scan.FAILED_INIT` or `5` `scan.SUCCESS` or `6`
scanCount	This is current number scans completed
stepCount	This is current number steps completed
reading	If measurements are taken during the scan, this parameter contains the last scan reading completed
bufferVar	A reading buffer used during scanning to store the readings. If a buffer is not specified, no readings are stored during the scan

Details

You can also use this function to specify the scanning reading buffer. This reading buffer, if specified, stores the readings and accompanying attributes as specified for the scan. An error is generated if the reading buffer does not exist or the parameter is not a reading buffer.

Before using this command, use `scan.create()` and `scan.add()`or `scan.addimagestep()` to set up a scan list.

When the scan is run in the background, you must use the `scan.state()` function to check the status of the scan.

Example

`scan.background(rbbuff1)`	Runs a scan in the background and stores readings in a buffer named `rbbuff1`.

Also see

scan.add() (on page 8-319)
scan.create() (on page 8-325)
scan.execute() (on page 8-327)
scan.list() (on page 8-328)
scan.state() (on page 8-336)
Scanning and triggering (on page 3-1)

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	Reset Scan reset Recall setup	Create configuration script Save setup	1 (scan.ON)

Usage

```
bypass = scan.bypass
scan.bypass = bypass
```

bypass	The state of the bypass. Set to one of the following values: `scan.OFF` or `0`: Disabled `scan.ON` or `1`: Enabled

Details

When *bypass* is `ON` and the `scan.trigger.arm.stimulus` is set to a non-zero value, the first channel of the scan closes (the `scan.trigger.channel.stimulus` setting is ignored).

For other channels (other than the first), the channel stimulus must be satisfied before the channel action takes place.

When *bypass* is `OFF`, every channel (including the first) must satisfy the `scan.trigger.channel.stimulus` setting before the channel action occurs for that step.

Example

`scan.bypass = scan.OFF` `print(scan.bypass)`	Disables the bypass option for scanning and displays the present bypass state. Output: `0.000000000e+000`

Also see

scan.trigger.arm.stimulus (on page 8-338)
scan.trigger.channel.stimulus (on page 8-341)
Scanning and triggering (on page 3-1)

scan.create()

This function deletes the existing scan list and creates a new list of channels and channel patterns to scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.create(channelList)
scan.create(channelList, dmmConfig)
```

channelList	String specifying channels to add
dmmConfig	The DMM configuration to use with items in the *channelList*

Details

The existing scan list is lost after calling this function.

The items in *channelList* are scanned in the order listed.

Each channel's (or channel pattern's) configuration is used unless the optional *dmmConfig* parameter is specified (see `dmm.setconfig()` and `dmm.getconfig()`). Specifying the *dmmConfig* parameter temporarily overrides the channel's or channel pattern's associated configuration. Specifying *dmmConfig* does not modify the assigned configuration of a channel or channel pattern.

If a forbidden channel is included in a range of channels or slot parameter (such as slot 1), the forbidden channel is ignored and no error is generated. If a forbidden channel is individually specified in the channel list, an error is generated.

You cannot specify an analog backplane relay as part of the channel list.

If an error occurs, the scan list of channels or channel patterns is cleared, even though no new scan list is created.

The function `scan.reset()` clears the list. To clear the scan list without performing a scan reset, send an empty string for the *channelList* parameter.

Example 1

`scan.create("1001:1010")`	Replaces the active scan list with an empty scan list. Adds channels 1 through 10 on slot 1. Uses the existing DMM configuration (`dmm.setconfig()`).

Example 2

```
scan.create()

for chan = 1001, 1010 do
   scan.add("" .. chan)


end
```

Replaces the active scan list with an empty scan list.

Loops through channels 1001 to 1010, and then adds 10 channels to the scan list. The parameter (`"" .. chan`) converts the channel number to a string.

The scan list now has, in order, channels 1 through 10 on slot 1.

Uses the existing DMM configuration (`dmm.setconfig()`).

Example 3

`scan.create("3001:3010", "testDCV")`	For this example, assume `testDCV` is a previously defined user DC volts configuration. Clears the old scan list and creates a new scan list with each channel (1 to 10 on slot 3). Each channel uses the DMM configuration associated with `testDCV`.

Example 4

```
scan.create("")


for chan = 3001, 3010 do

   scan.add("" .. chan, "testDCV")


   scan.add("" .. chan, "test2wire")

end
```

For this example, assume `testDCV` is a previously defined user DC volts configuration, and `test2wire` is a previously defined user 2-wire ohm configuration.
This loops through the channels, adding channels to the scan list. The first time, it adds `"testDCV"` for a channel. The second time, it adds `"test2wire"` for that channel. The first parameter `("" .. chan)` converts the chan number to a string.
Clears the old scan list and creates a new scan list.
Loops through channels 3001 to 3010.
Adds channels 3001 through 3010 to the end of the existing scan list.

Also see

dmm.getconfig() (on page 8-189)
dmm.setconfig() (on page 8-237)
scan.add() (on page 8-319)
scan.reset() (on page 8-334)
Scanning and triggering (on page 3-1)

scan.execute()

This function starts the scan immediately in the foreground with a configured scan list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
state, scanCount, stepCount, reading = scan.execute()
state, scanCount, stepCount, reading = scan.execute(bufferVar)
```

state	The result of scanning: `scan.EMPTY` or `0` `scan.BUILDING` or 1 `scan.RUNNING` or 2 `scan.ABORTED` or 3 `scan.FAILED` or 4 `scan.FAILED_INIT` or 5 `scan.SUCCESS` or 6
scanCount	The present number of scans completed
stepCount	The present number of steps completed
reading	If measurements are taken during the scan, this parameter contains the last scan reading completed
bufferVar	A reading buffer used during scanning to store the readings. If a buffer is not specified, no readings are stored during the scan

Details

In addition to starting and running the scan in immediate mode (not in the background), you can use this function to specify the scanning reading buffer. This reading buffer stores the readings and accompanying attributes as specified for the scan. An error is generated if the reading buffer does not exist or if the parameter is not a reading buffer.

Before using this command, use `scan.create()` and `scan.add()` or `scan.addimagestep()` to set up a scan list.

Execution runs until the scan is complete or until the `abort` command is sent.

Because this function waits for the scan to complete, the `scan.state()` function cannot be used to see the current status of scanning.

Example

`scan.execute(rbBuff1)`	Runs a scan immediately and stores the readings in a reading buffer named `rbbuff1`.

Also see

scan.add() (on page 8-319)
scan.background() (on page 8-323)
scan.create() (on page 8-325)
scan.list() (on page 8-328)
scan.state() (on page 8-336)
Scanning and triggering (on page 3-1)

scan.list()

This function queries the active scan list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	Instrument reset Channel reset Scan reset Recall setup Change of channel or scan setting	Create configuration script Save setup	Empty list

Usage

```
scanList = scan.list()
```

scanList	A string that lists the existing scan step information

Details

This function lists the existing scan list.

When you change a channel or scan attribute for an existing scan list item, the scan list is recreated based on this change. If the scan list cannot be rebuilt, an error is generated and the scan list is lost.

To avoid unintentional changes to an existing scan list, configure channel and scan settings before using the commands `scan.add()`, `scan.addimagestep()`, and `scan.create()`) to build a scan list.

If the scan list is empty, the string `"Empty Scan"` is returned. Otherwise, the string lists each step in the scan along with its information for step, open, measure configuration, count, and close (see the example below).

Example

`reset()` `dmm.setconfig("2020, 2021", "dcvolts")` `dmm.nplc = 0.5` `dmm.range = 10` `dmm.configure.set("DCVSlot2")` `dmm.setconfig("2016,2017", "DCVSlot2")` `scan.create("2007,2008,2020,2021,2016,2017")` `print(scan.list())`	Assume a Model 3721 in slot 2. Configure channels 20 and 21 for DC volts on slot 2. Change the DMM settings for NPLC and range and save those DC volt settings as `"DCVSlot2"`. Configure channels 16 and 17 for `"DCVSlot2"` on slot 2. Populate the scan list with the function `scan.create("`2007,2008,2020,2021,2016,2017`")`, then initiate the scan list to be output. Outputs the existing scan list. For example, an existing scan list may appear as follows: `Init) OPEN...` `1) STEP: 2007` `CLOSE: 2007` `MEASURE: nofunction COUNT: 1` `2) STEP: 2008` `OPEN: 2007` `CLOSE: 2008` `MEASURE: nofunction COUNT: 1` `3) STEP: 2020` `OPEN: 2008` `CLOSE: 2020 2911` `MEASURE: dcvolts COUNT: 1` `4) STEP: 2021` `OPEN: 2020 2911` `CLOSE: 2021 2921` `MEASURE: dcvolts COUNT: 1`
	`5) STEP: 2016` `OPEN: 2021 2921` `CLOSE: 2016 2911` `MEASURE: DCVSlot2 COUNT: 1` `6) STEP: 2017` `OPEN: 2016` `CLOSE: 2017` `MEASURE: DCVSlot2 COUNT: 1`

Also see

scan.create() (on page 8-325)
Scanning and triggering (on page 3-1)

scan.measurecount

This attribute sets the number of iterations performed when a scanning measurement is requested.

Type	TSP-Link™ accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Scan reset Recall setup	Create configuration script Save setup	1

Usage

```
count = scan.measurecount
scan.measurecount = count
```

count	The count value being used or read; valid range is 1 to 450000

Details

Use this attribute to indicate how many measurements to make on a step when measurements are needed. This sets the measurement count in the trigger model. During a scan, the Model 3706A iterates through the sequence event detector and measurement action of the trigger model *count* times. After performing *count* iterations, the Model 3706A returns to check the scan count.

This must be set before the scan is started. Once set, it applies to all scan steps in the list, including scan steps that exist in the list and any that are added before the scan is started.

All steps take the same number of measurements. When taking multiple measurements, the measurements may be taken as quickly as possible based on the configuration (`scan.trigger.measure.stimulus = 0`) or they may be paced by a trigger (`scan.trigger.measure.stimulus` is nonzero).

Example

`scan.measurecount = 5`	Sets the measurement count to 5.

Also see

scan.create() (on page 8-325)
Scanning and triggering (on page 3-1)

scan.mode

This attribute controls the scan mode setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Scan reset Recall setup	Create configuration script Save setup	0 (scan.MODE_OPEN_ALL)

Usage

```
scanModeSetting = scan.mode
scan.mode = scanModeSetting
```

scanModeSetting	The present scan mode setting. Set to one of the following values: • `scan.MODE_OPEN_ALL` or `0` • `scan.MODE_OPEN_SELECTIVE` or 1: See Details • `scan.MODE_FIXED_ABR` or 2: See Details

Details

When this attribute is set to `scan.MODE_OPEN_ALL`, all channels on all slots are opened before a scan starts.

When this attribute is set to `scan.MODE_OPEN_SELECTIVE`, an intelligent open is performed. Assuming all steps being scanned have a function value of "`nofunction`" with their DMM configuration then:

- All channels and analog backplane relays involved in scanning are opened
- Closed channels and backplane relays not involved in scanning remain closed during the scan

If any step has a DMM configuration with a function set to any other value than "`nofunction`":

- Analog backplane relays 1 and 2 are opened on all slots
- 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 `scan.MODE_FIXED_ABR`, it is equivalent to setting `MODE_OPEN_SELECTIVE` , except:

- All required backplane relays are closed before the start of the scan
- These backplane relays are not opened or closed during the scan
- These backplane relays do not open at the end of scan

Example

`scan.mode = scan.MODE_OPEN_SELECTIVE`	Sets the scan mode setting to open selective.

Also see

scan.reset() (on page 8-334)
Scanning and triggering (on page 3-1)

scan.nobufferbackground()

This function starts a scan in background mode and specifies that no reading buffer is used during scanning.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`state, scancount, stepcount = scan.nobufferbackground()`

state	The result of scanning: • `scan.EMPTY` or `0` • `scan.BUILDING` or 1 • `scan.RUNNING` or 2 • `scan.ABORTED` or 3 • `scan.FAILED` or 4 • `scan.FAILED_INIT` or 5 • `scan.SUCCESS` or 6
scancount	The present number of scans completed
stepcount	The present number of steps completed

Details

Before using this command, use `scan.create()`, `scan.add()` and `scan.addimagestep()` to set up scan elements. If a reading buffer is specified, an error is generated.

To view the scan status, use `scan.state()`.

To run a scan in the background with a reading buffer, see scan.background() (on page 8-323).

Example

`scan.nobufferbackground()`	Run the scan in the background with no reading buffer.

Also see

scan.add() (on page 8-319)
scan.background() (on page 8-323)
scan.create() (on page 8-325)
scan.execute() (on page 8-327)
scan.list() (on page 8-328)
scan.nobufferexecute() (on page 8-333)
scan.state() (on page 8-336)
Scanning and triggering (on page 3-1)

scan.nobufferexecute()

This function starts a scan immediately and specifies that no reading buffer is used during scanning.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`state, scanCount, stepCount = scan.nobufferbackground()`

state	The result of scanning: • `scan.EMPTY` or `0` • `scan.BUILDING` or 1 • `scan.RUNNING` or 2 • `scan.ABORTED` or 3 • `scan.FAILED` or 4 • `scan.FAILED_INIT` or 5 • `scan.SUCCESS` or 6
scanCount	The present number of scans that have completed
stepCount	The present number of steps have completed

Details

Before using this command, use `scan.create()`, `scan.add()`, and `scan.addimagestep()` to set up scan elements. If a reading buffer is specified, an error is generated.

The command continues execution until scanning completes or is aborted by the user.

To run a scan immediately with a reading buffer, see scan.execute() (on page 8-327).

Example

`scan.nobufferexecute()`	Runs the scan immediately with no reading buffer.

Also see

scan.add() (on page 8-319)
scan.background() (on page 8-323)
scan.create() (on page 8-325)
scan.execute() (on page 8-327)
scan.list() (on page 8-328)
scan.nobufferbackground() (on page 8-332)
scan.state() (on page 8-336)
Scanning and triggering (on page 3-1)

scan.reset()

This function resets the trigger model and scan list settings to their factory default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.reset()
```

Details

When `scan.reset()` is sent, the trigger model and scan settings that are reset to the factory defaults are:

- `scan.bypass`
- `scan.measurecount`
- `scan.mode`
- `scan.scancount`
- `scan.trigger.arm.stimulus`
- `scan.trigger.channel.stimulus`
- `scan.trigger.measure.stimulus`
- `scan.trigger.sequence.stimulus`

In addition, the scan list is cleared.

NOTE

Sending this function only affects the trigger model and scan list settings. To reset all instrument settings to factory default settings, use the `reset()` command.

Example

`scan.reset()`	Performs a reset on the trigger model and scan settings.

Also see

channel.reset() (on page 8-87)
dmm.reset() (on page 8-228)
reset() (on page 8-317)
scan.bypass (on page 8-324)
scan.measurecount (on page 8-330)
scan.mode (on page 8-331)
scan.scancount (on page 8-335)
scan.trigger.arm.stimulus (on page 8-338)
scan.trigger.channel.stimulus (on page 8-341)
scan.trigger.measure.stimulus (on page 8-345)
scan.trigger.sequence.stimulus (on page 8-348)
Scanning and triggering (on page 3-1)

scan.scancount

This attribute sets the scan count in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Scan reset Recall setup	Create configuration script Save setup	1

Usage

```
scanCount = scan.scancount
scan.scancount = scanCount
```

scanCount	The present scan count value (1 to 2,000,000,000)

Details

The scan count attribute setting indicates how many times the scan list is iterated through before the scan completes.

During a scan, the instrument iterates through the arm layer of the trigger model the specified number of times. After performing the specified number of iterations, the instrument returns to an idle state.

Example

`scan.scancount = 5`	Sets the scan count to 5.

Also see

Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

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, reading = scan.state()`

scanState	The present state of the scan running in the background. Possible states include: `scan.EMPTY` or `0` `scan.BUILDING` or 1 `scan.RUNNING` or 2 `scan.ABORTED` or 3 `scan.FAILED` or 4 `scan.FAILED_INIT` or 5 `scan.SUCCESS` or 6
scanCount	The current number of scans that have completed
stepCount	The current number of steps that have completed
reading	If measurements are taken during the scan, this parameter contains the last scan reading completed

Details

scanCount is the number of the current 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 (0).

stepCount 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.

Example

`scan.background()` `scanState, scanCount, stepCount = scan.state()` `print(scanState)`	Runs a scan in the background. Check the present scan state. View value of `scanState`. Output shows that scan is running: `2.00000e+00`

Also see

scan.background() (on page 8-323)
scan.mode (on page 8-331)
Scanning and triggering (on page 3-1)

scan.stepcount

This attribute contains 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
```

scanStepCount	The present step count value

Details

This is set by the number of steps in the active scan list. The value of this attribute is initially determined when the scan is created. Adding steps with the `scan.create(), scan.addimagestep()`, and `scan.add()` functions updates this attribute's value.

Example

`print(scan.stepcount)`	Responds with the present step count. Output assuming there are five steps in the scan list: `5.00000e+00`

Also see

scan.add() (on page 8-319)
scan.addimagestep() (on page 8-321)
scan.create() (on page 8-325)
Scanning and triggering (on page 3-1)

scan.trigger.arm.clear()

This function clears the arm event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.arm.clear()
```

Details

This function sets the trigger model's arm event detector to the undetected state.

Example

`scan.trigger.arm.clear()`	Clears the arm event detector.

Also see

scan.trigger.arm.set() (on page 8-338)
scan.trigger.arm.stimulus (on page 8-338)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.arm.set()

This function sets the arm event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.arm.set()
```

Details

This function sets the arm event detector of the trigger model to the detected state.

Example

`scan.trigger.arm.set()`	Sets the arm event detector to the detected state.

Also see

scan.trigger.arm.clear() (on page 8-337)
scan.trigger.arm.stimulus (on page 8-338)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.arm.stimulus

This attribute determines which event starts the scan.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Scan reset	Create configuration script Save setup	0

Usage

```
eventID = scan.trigger.arm.stimulus
scan.trigger.arm.stimulus = eventID
```

eventID	Trigger stimulus used for the channel action (arm layer); see **Details**

Details

This attribute selects which events cause the arm event detector to enter the detected state.

Set this attribute to 0 to start the scan without waiting for an event.

eventID may be one of the following trigger event IDs.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID or 41 to 48`	The trigger event generated by the channel trigger *N*.
`digio.trigger[N].EVENT_ID or 1 to 14`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID or 39`	The trigger key (TRIG) on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH or 53`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW or 52`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH or 55`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW or 54`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID or 40`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID or 58 to 59`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID or 20 to 23`	A delay expired.
`tsplink.trigger[N].EVENT_ID or 17 to 19`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID or 29 to 36`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY or 24`	Scan ready event.
`scan.trigger.EVENT_SCAN_START or 25`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY or 28`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP or 56`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP or 50`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP or 26`	Scan complete event.
`scan.trigger.EVENT_IDLE or 27`	Idle event.
`schedule.alarm[N].EVENT_ID or 37 to 38`	Trigger event generated by the alarm *N*.

NOTE

Use one of the text trigger event IDs (for example, `digio.trigger[N].EVENT_ID`) to set the stimulus value rather than the numeric value. Doing this will make the code compatible for future upgrades.

Example 1

`scan.trigger.arm.stimulus = scan.trigger.EVENT_SCAN_READY`	Sets trigger stimulus of the arm event detector to scan ready event.

Example 2

`scan.trigger.arm.stimulus = 0`	The scan begins immediately.

Example 3

`scan.trigger.arm.stimulus = digio.trigger[3].EVENT_ID`	The scan begins when the instrument receives a signal from digital I/O line 3.

Also see

scan.trigger.arm.clear() (on page 8-337)
scan.trigger.arm.set() (on page 8-338)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.channel.clear()

This function clears the channel event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.channel.clear()
```

Details

This function clears the channel event detector of the trigger model (sets it to the undetected state).

Example

`scan.trigger.channel.clear()`	Clears the channel event detector.

Also see

scan.trigger.channel.set() (on page 8-341)
scan.trigger.channel.stimulus (on page 8-341)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.channel.set()

This function sets the channel event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.channel.set()
```

Details

This function sets the channel event detector of the trigger model to the detected state.

Example

`scan.trigger.channel.set()`	Sets the channel event detector of the trigger model to the detected state.

Also see

scan.trigger.channel.clear() (on page 8-340)
scan.trigger.channel.stimulus (on page 8-341)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.channel.stimulus

This attribute determines which trigger events cause the channel actions to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Scan reset	Create configuration script Save setup	50 (scan.trigger.EVENT_SEQUENCE_COMP)

Usage

```
eventID = scan.trigger.channel.stimulus
scan.trigger.channel.stimulus = eventID
```

eventID	Trigger stimulus used for the channel action; see **Details** for possible trigger event IDs

Details

This attribute selects which events cause the channel event detector to enter the detected state. Set this attribute to `0` to start the channel action immediately at the default setting.

Set *`eventID`* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID or 41 to 48`	The trigger event generated by the channel trigger *N*.
`digio.trigger[N].EVENT_ID or 1 to 14`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID or 39`	The trigger key (TRIG) on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH or 53`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW or 52`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH or 55`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW or 54`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID or 40`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID or 58 to 59`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID or 20 to 23`	A delay expired.
`tsplink.trigger[N].EVENT_ID or 17 to 19`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID or 29 to 36`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY or 24`	Scan ready event.
`scan.trigger.EVENT_SCAN_START or 25`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY or 28`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP or 56`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP or 50`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP or 26`	Scan complete event.
`scan.trigger.EVENT_IDLE or 27`	Idle event.
`schedule.alarm[N].EVENT_ID or 37 to 38`	Trigger event generated by the alarm *N*.

NOTE

Use one of the text trigger event IDs (for example, `digio.trigger[N].EVENT_ID`) to set the stimulus value rather than the numeric value. Doing this will make the code compatible for future upgrades.

Example 1

`scan.trigger.channel.stimulus = scan.trigger.EVENT_SCAN_START`	Sets the trigger stimulus of the channel event detector to scan start event.

Example 2

`scan.trigger.channel.stimulus = 0` `print(scan.trigger.channel.stimulus)`	Starts the channel action immediately after the Scan Start Event. This also resets the stimulus to the default. Output: `5.000000000e+01`

Also see

scan.trigger.channel.clear() (on page 8-340)
scan.trigger.channel.set() (on page 8-341)
Trigger model (on page 3-2)
Scanning and triggering (on page 3-1)

scan.trigger.clear()

This function clears the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.clear()
```

Details

This function sets the arm, channel, measurement, and sequence event detectors of the trigger model to the undetected state.

Example

`scan.trigger.clear()`	Clears the trigger model.

Also see

scan.trigger.channel.set() (on page 8-341)
scan.trigger.channel.stimulus (on page 8-341)
Scanning and triggering (on page 3-1)
Trigger model (on page 3-2)

scan.trigger.measure.clear()

This function clears the measure event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.measure.clear()
```

Details

This function sets the measurement event detector of the trigger model to the undetected state.

Example

`scan.trigger.measure.clear()`	Clears the measurement event detector.

Also see

Scanning and triggering (on page 3-1)

scan.trigger.measure.set()

This function sets the measurement event detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.measure.set()
```

Details

This function sets the measurement event detector of the trigger model to the detected state.

Example

`scan.trigger.measure.set()`	Sets the measurement event detector to the detected state.

Also see

Scanning and triggering (on page 3-1)

scan.trigger.measure.stimulus

This attribute selects the trigger stimulus of the event detector trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Scan reset	Create configuration script Save setup	0

Usage

```
eventID = scan.trigger.measure.stimulus
scan.trigger.measure.stimulus = eventID
```

eventID	The trigger stimulus that is used for the measurement event

Details

This attribute selects the events that cause the measurement event detector to enter the detected state. You can use this to pace each one of the measurement count readings with an event.

To pace all readings by a single event, use `scan.trigger.sequence.stimulus`.

To bypass waiting for an event, set this to `0`.

eventID can be set to one of the existing trigger event IDs, shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID or 41 to 48`	The trigger event generated by the channel trigger *N*.
`digio.trigger[N].EVENT_ID or 1 to 14`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID or 39`	The trigger key (TRIG) on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH or 53`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW or 52`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH or 55`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW or 54`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID or 40`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID or 58 to 59`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID or 20 to 23`	A delay expired.
`tsplink.trigger[N].EVENT_ID or 17 to 19`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID or 29 to 36`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY or 24`	Scan ready event.
`scan.trigger.EVENT_SCAN_START or 25`	Scan start event.

`scan.trigger.EVENT_CHANNEL_READY or 28`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP or 56`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP or 50`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP or 26`	Scan complete event.
`scan.trigger.EVENT_IDLE or 27`	Idle event.
`schedule.alarm[N].EVENT_ID or 37 to 38`	Trigger event generated by the alarm *N*.

NOTE

Use one of the text trigger event IDs (for example, `digio.trigger[N].EVENT_ID`) to set the stimulus value rather than the numeric value. Doing this will make the code compatible for future upgrades.

Example

`scan.trigger.measure.stimulus = scan.trigger.EVENT_CHANNEL_READY`	
Sets the trigger stimulus of the measurement event detector to the channel ready event.	

Also see

scan.trigger.sequence.stimulus (on page 8-348)
Scanning and triggering (on page 3-1)

scan.trigger.sequence.clear()

This function clears the sequence event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.sequence.clear()
```

Details

This function sets the sequence event detector to the undetected state.

Example

`scan.trigger.sequence.clear()`	Clears the sequence event detector.

Also see

Scanning and triggering (on page 3-1)

scan.trigger.sequence.set()

This function sets the sequence even detector to the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
scan.trigger.sequence.set()
```

Details

This function sets the sequence event detector to the detected state.

Example

`scan.trigger.sequence.set()`	Sets the sequence event detector to the detected state.

Also see

scan.trigger.sequence.clear() (on page 8-346)
Scanning and triggering (on page 3-1)

scan.trigger.sequence.stimulus

This attribute selects the trigger stimulus for the sequence event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Scan reset	Create configuration script Save setup	28 (scan.trigger.EVENT_CHANNEL_READY)

Usage

```
eventID = scan.trigger.sequence.stimulus
scan.trigger.sequence.stimulus = eventID
```

eventID	The trigger stimulus that is used for the sequence event

Details

This attribute selects the events that cause the sequence event detector to enter the detected state.

Use this to start a set of measurement count readings that are triggered by a single event.

To pace each reading by an event, use `scan.trigger.measure.stimulus`.

To bypass pacing the readings, set this to 0.

Set *eventID* to one of the existing trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
channel.trigger[N].EVENT_ID or 41 to 48	The trigger event generated by the channel trigger *N*.
digio.trigger[N].EVENT_ID or 1 to 14	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
display.trigger.EVENT_ID or 39	The trigger key (TRIG) on the front panel is pressed.
dmm.trigger.EVENT_LIMIT1_HIGH or 53	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
dmm.trigger.EVENT_LIMIT1_LOW or 52	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
dmm.trigger.EVENT_LIMIT2_HIGH or 55	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
dmm.trigger.EVENT_LIMIT2_LOW or 54	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
trigger.EVENT_ID or 40	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
trigger.blender[N].EVENT_ID or 58 to 59	A combination of events has occurred.
trigger.timer[N].EVENT_ID or 20 to 23	A delay expired.
tsplink.trigger[N].EVENT_ID or 17 to 19	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
lan.trigger[N].EVENT_ID or 29 to 36	A LAN trigger event has occurred.
scan.trigger.EVENT_SCAN_READY or 24	Scan ready event.
scan.trigger.EVENT_SCAN_START or 25	Scan start event.

`scan.trigger.EVENT_CHANNEL_READY or 28`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP or 56`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP or 50`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP or 26`	Scan complete event.
`scan.trigger.EVENT_IDLE or 27`	Idle event.
`schedule.alarm[N].EVENT_ID or 37 to 38`	Trigger event generated by the alarm *N*.

NOTE

Use one of the text trigger event IDs (for example, `digio.trigger[N].EVENT_ID`) to set the stimulus value rather than the numeric value. Doing this will make the code compatible for future upgrades.

Example

```
scan.trigger.sequence.stimulus = scan.trigger.EVENT_CHANNEL_READY
```

Sets the trigger stimulus of the sequence event detector to the channel ready event.

Also see

scan.trigger.measure.stimulus (on page 8-345)
Scanning and triggering (on page 3-1)

schedule.alarm[N].enable

This attribute enables or disables an alarm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup	NOt saved	0 (schedule.OFF)

Usage

state = schedule.alarm[*N*].enable
schedule.alarm[*N*].enable = *state*

state	Disable the alarm(schedule.OFF or 0) Enable the alarm (schedule.ON or 1)
N	Alarm number (1 or 2)

Details

If you enable an alarm that has a start time that is in the past, the alarm executes immediately.

If an alarm time in the past is used to start a scan, the alarm time may be missed by the scan start. This can occur because the scan clears any pending triggers before it begins, so it will miss any trigger generated from the alarm enable. To prevent a missed alarm, start the scan in the background, then enable the alarm.

Example

schedule.alarm[1].enable = 1	Enables alarm 1.

Also see

schedule.alarm[N].EVENT_ID (on page 8-351)
schedule.alarm[N].fractionalseconds (on page 8-352)
schedule.alarm[N].period (on page 8-353)
schedule.alarm[N].ptpseconds (on page 8-353)
schedule.alarm[N].repetition (on page 8-354)
schedule.alarm[N].seconds (on page 8-355)
schedule.disable() (on page 8-355)

schedule.alarm[N].EVENT_ID

This constant describes the trigger event generated by the alarm *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = schedule.alarm[*N*].EVENT_ID

eventID	The trigger event number
N	Alarm number (1 or 2)

Details

To have another trigger object respond to trigger events generated by the schedule alarm, set the other object's stimulus attribute to the value of this constant.

Example

`scan.trigger.arm.stimulus = schedule.alarm[1].EVENT_ID`	Uses a trigger event on alarm 1 to be the stimulus for the trigger arm.

Also see

None

schedule.alarm[N].fractionalseconds

This attribute describes the fractional seconds portion of the alarm time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset	Not saved	0

Usage

```
schedule.alarm[N].fractionalseconds = fraction
fraction = schedule.alarm[N].fractionalseconds
```

N	Alarm number (1 or 2)
fraction	The fractional seconds portion of the alarm time

Details

1588 has too much resolution to represent in a single floating point value so the alarm times are split into two values (seconds and fractional seconds).

Example

```
-- get current time and store in variable sec
sec = os.time()
-- set alarm 1 seconds to be 1 minute after current time
schedule.alarm[1].seconds = sec + 60
-- set alarm 1 fractional seconds to be 0.5
schedule.alarm[1].fractionalseconds = 0.5
print("value of sec is ", sec)
print("value of alarm 1 seconds is ",  schedule.alarm[1].seconds)
print("value of alarm 1 fractional seconds is ",
   schedule.alarm[1].fractionalseconds)
```

Create an alarm to occur 60.5 seconds from current time in UTC seconds.

Output:

```
value of sec is    1.306405866e+009
value of alarm 1 seconds is     1.306405926e+009
value of alarm 1 fractional seconds is 5.000000000e-001
```

Also see

schedule.alarm[N].seconds (on page 8-355)

schedule.alarm[N].period

This attribute describes the time, in seconds, between adjacent firings of the alarm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup	Create configuration script Save setup	0

Usage

```
value = schedule.alarm[N].period
schedule.alarm[N].period = value
```

N	Alarm number (1 or 2)
value	The time in seconds

Example

`schedule.alarm[1].period = 0.5`	Set a period of 0.5 seconds between firings of alarms after the initial alarm.

Also see

None

schedule.alarm[N].ptpseconds

The seconds portion of the alarm time in PTP seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset	Not saved	0

Usage

```
schedule.alarm[N].ptpseconds = seconds
seconds = schedule.alarm[N].ptpseconds
```

N	Alarm (1 or 2)
seconds	The seconds portion of the alarm time in PTP seconds

Details

1588 has too much resolution to represent in a single floating point value, so the alarm times are split into two values (seconds and fractional seconds).

Example

`sec,ns = ptp.time()` `schedule.alarm[1].ptpseconds = sec + 30`	Create an alarm to occur 30 seconds from current time in PTP seconds.

Also see

ptp.utcoffset (on page 8-317)

schedule.alarm[N].repetition

This attribute describes the number of times an alarm repeats after the first alarm firing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup	Save setup	0

Usage

```
count = schedule.alarm[N].repetition
schedule.alarm[N].repetition = count
```

count	The number of repetitions
N	Alarm 1 or 2

Details

The alarm will fire a total of `count+1` times. If 0 and period is non-zero, the alarm fires forever.

Once an alarm begins, the repetition counts down for each trigger generated. It ends at zero (0). You must set this repetition back to some value if you intend to reissue the alarm. Otherwise, the alarm will either not fire (if the period is zero) or fire forever (if period is non-zero).

Example

`schedule.alarm[1].repetition = 10`	Set the alarm to fire 10 times.

Also see

schedule.alarm[N].enable (on page 8-350)

schedule.alarm[N].seconds

The seconds portion of the alarm time in UTC seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset	Not saved	0

Usage

```
value = schedule.alarm[N].seconds
schedule.alarm[N].seconds = value
```

value	Seconds portion of the alarm time in UTC seconds
N	Alarm number (1 or 2)

Details

1588 has too much resolution to represent in a single floating point value, so the alarm times are split into two values (seconds and fractional seconds).

Example

```
local l_myTime
l_myTime = os.time{year = 2008, month = 3, day = 15, hour = 10}
schedule.alarm[1].seconds = l_myTime
```

Create an alarm to occur on March 15, 2008 at 10 am in UTC seconds.

Also see

ptp.utcoffset (on page 8-317)

schedule.disable()

This function disables all alarms.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
schedule.disable()
```

Details

This command sets the `schedule.alarm[N].enable` attribute to 0 (`schedule.OFF`) for each schedule alarm *N*.

Also see

schedule.alarm[N].enable (on page 8-350)

script.anonymous

This is a reference to the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	See **Details**	See **Details**	Not applicable

Usage

```
scriptVar = script.anonymous
```

scriptVar	The name of the variable that references the script

Details

You can use the `script.anonymous` script like any other script. Also, you can save the anonymous script as a user script by giving it a name.

This script is replaced by loading a script with the `loadscript` or `loadandrunscript` commands when they are used without a name.

Example 1

`script.anonymous.list()`	Displays the content of the anonymous script.

Example 2

`print(script.anonymous.source)`	Retrieves the source of the anonymous script.

Also see

Anonymous scripts (on page 7-3)
scriptVar.autorun (on page 8-361)
scriptVar.list() (on page 8-362)
scriptVar.name (on page 8-363)
scriptVar.run() (on page 8-364)
scriptVar.save() (on page 8-365)
scriptVar.source (on page 8-365)

script.delete()

This function deletes a script from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.delete(scriptName)
```

scriptName	The string that represents the name of the script

Example

`script.delete("test8")`	Deletes a user script named "`test8`" from nonvolatile memory.

Also see

Delete user scripts (on page 7-10)
Delete user scripts from the instrument (on page 7-43)
scriptVar.save() (on page 8-365)

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

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

scriptVar	The created script; this is `nil` if an error is encountered
file	The path and file name of the script file to load
name	The name of the script to be created

Details

The file path may be absolute or relative to the current working directory. The root folder of the USB flash drive has the absolute path "/usb1/". Both the forward slash (/) and backslash (\) are supported as directory separators.

The file to be loaded must start with the `loadscript` or `loadandrunscript` keywords, contain the body of the script, and end with the `endscript` keyword.

Script naming:

- If the *name* parameter is an empty string, or *name* is absent (or `nil`) and the script name cannot be extracted from the file, *scriptVar* is the only handle to the created script.
- If *name* is given (and not `nil`), any script name embedded in the file is ignored.
- If *name* conflicts with the name of an existing script in the `script.user.scripts` table, the existing script's name attribute is set to an empty string before it is replaced in the `script.user.scripts` table by the new script.
- If *name* is absent or `nil`, the command attempts to extract the name of the script from the file. Any conflict between the extracted name and that of an existing script in the scripts table generates an error. If the script name cannot be extracted, the created script's name attribute is initialized to the empty string, and must be set to a valid nonempty string before saving the script to nonvolatile memory.

Example

`myTest8 = script.load("/usb1/filename.tsp", "myTest8")`	Loads the script myTest8 from the USB flash drive.

Also see

script.new() (on page 8-358)

script.new()

This function creates a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

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

scriptVar	The name of the variable that will reference the script
code	A string containing the body of the script
name	The name of the script

Details

The *name* parameter is the name that is added to the `script.user.scripts` table. If *name* is not given, an empty string will be used, and the script will be unnamed. If the name already exists in `script.user.scripts`, the existing script's `name` attribute is set to an empty string before it is replaced by the new script.

Note that *name* is the value that is used for the instrument front panel display. If this value is not defined, the script will not be available from the instrument front panel.

You must save the new script into nonvolatile memory to keep it when the instrument is turned off.

Example 1

```
myTest8 = script.new(
   "display.clear() display.settext('Hello from myTest8')", "myTest8")
myTest8()
```

Creates a new script referenced by the variable `myTest8` with the name "`myTest8`".
Runs the script. The instrument displays "`Hello from myTest8`".

Example 2

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

Creates a new autoexec script that clears the display when the instrument is turned on and displays "`Hello from autoexec`".

Also see

Create a script using the script.new() command (on page 7-38)
Global variables and the script.user.scripts table (on page 7-36)
Named scripts (on page 7-4)
scriptVar.save() (on page 8-365)
script.newautorun() (on page 8-359)

script.newautorun()

This function is identical to the `script.new()` function, but it creates a script with the autorun attribute set to `"yes"`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.newautorun(code)
scriptVar = script.newautorun(code, name)
```

scriptVar	The name of the variable that will reference the script
code	A string containing the body of the script
name	The name of the script

Details

The `script.newautorun()` function is identical to the `script.new()`function, except that the autorun attribute of the script is set to `yes`. The script is also automatically run immediately after it is created.

Example

```
NewAuto = script.newautorun("print('Hello from new auto run command')",
   'NewAuto')
print(NewAuto.autorun)
print(NewAuto.name)
```

Creates a new script called `NewAuto` that automatically has the autorun attribute set to yes after it is created. The *name* attribute's value is set to `"NewAuto"`.

Output:

```
Hello from new auto run command
yes
NewAuto
```

Also see

Create a script using the script.new() command (on page 7-38)
Global variables and the script.user.scripts table (on page 7-36)
Named scripts (on page 7-4)
script.new() (on page 8-358)
scriptVar.save() (on page 8-365)

script.restore()

This function restores a script that was removed from the run-time environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.restore(name)
```

name	The name of the script to be restored

Details

This command copies the script from nonvolatile memory into the run-time environment. It also creates a global variable with the same name as the name of the script.

Example

`script.restore("test9")`	Restores a script named "test9" from nonvolatile memory.

Also see

script.delete() (on page 8-356)

script.run()

This function runs the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

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

Details

Each time the `script.run()` command is given, the anonymous script is executed. This script can be run using this command many times without having to re-send it.

Example

`run()`	Runs the anonymous script.

Also see

script.anonymous (on page 8-355)

script.user.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.user.catalog() do body end`

name	String representing the name of the script
body	Code that implements the body of the `for` loop to process the names in the catalog

Details

Accessing the catalog of scripts stored in nonvolatile memory allows you to process all scripts in nonvolatile memory. The entries will be 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

`for name in script.user.catalog() do` `   print(name)` `end`	Retrieve the catalog listing for user scripts.

Also see

None

scriptVar.autorun

This attribute controls the autorun state of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	See **Details**	See **Details**

Usage

```
scriptVar.autorun = state
state = scriptVar.autorun
```

scriptVar	The name of the variable that references the script
state	Whether or not the script runs automatically when powered on: • `"yes"` (script runs automatically) • `"no"` (script does not run automatically)

Details

Autorun scripts run automatically when the instrument is turned on. You can set any number of scripts to autorun.

The run order for autorun scripts is arbitrary, so make sure the run order is not important.

The default value for *`scriptVar`*`.autorun` depends on how the script was loaded. The default is `"no"` if the script was loaded with `loadscript` or `script.new()`. It is `"yes"` for scripts loaded with `loadandrunscript` or `script.newautorun()`.

NOTE

Make sure to save the script in nonvolatile memory after setting the `autorun` attribute so that the instrument will retain the setting.

Example

`test5.autorun = "yes"` `test5.save()`	Assume a script named "test5" is in the run-time environment. The next time the instrument is turned on, "test5" script automatically loads and runs.

Also see

None

scriptVar.list()

This function generates a script listing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

scriptVar.list()

scriptVar	The name of variable that references the script

Details

This function generates output in the form of a sequence of response messages (one message for each line of the script). It also generates output of the script control messages (`loadscript` or `loadandrunscript` and `endscript`).

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')",
    "test7")
test7()
test7.save()
test7.list()
```

The above example code creates a script named "test7" that displays text on the front panel and lists the script with the following output:

```
loadscript test7
display.clear() display.settext("Hello from my test")
endscript
```

Also see

Load a script by sending commands over the remote interface (on page 7-4)
Retrieve source code one line at a time (on page 7-8)

scriptVar.name

This attribute contains the name of a script in the run-time environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	Not applicable

Usage

```
scriptVar.name = scriptName
scriptName = scriptVar.name
```

scriptVar	Name of the variable that references the script
scriptName	A string that represents the name of the script

Details

When setting the script name, this attribute renames the script that the variable `scriptVar` references.

This attribute must be either a valid Lua identifier or the empty string. Changing the name of a script changes the index that is used to access the script in the `script.user.scripts` table. Setting the attribute to an empty string removes the script from the table completely, and the script becomes an unnamed script.

As long as there are variables referencing an unnamed script, the script can be accessed through those variables. When all variables that reference an unnamed script are removed, the script will be removed from the run-time environment.

If the new name is the same as a name that is already used for another script, the name of the other script is set to an empty string, and that script becomes unnamed.

NOTE

Changing the name of a script does not change the name of any variables that reference that script. The variables will still reference the script, but the names of the script and variables may not match.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
test7()
print(test7.name)

test7.name = "test7"
print(test7.name)

test7.save()
```

This example calls the `script.new()` function to create a script with no name, runs the script, names the script "`test7`", and then saves the script in nonvolatile memory.

Also see

Rename a script (on page 7-41)
script.new() (on page 8-358)
scriptVar.save() (on page 8-365)

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

scriptVar.run()
scriptVar()

scriptVar	The name of the variable that references the script

Details

The *scriptVar*.run() function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar*().

Example

test8.run()	Runs the script referenced by the variable test8.

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory or to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()
scriptVar.save(filename)
```

scriptVar	The name of variable that references the script
filename	The file name to use when saving the script to a USB flash drive

Details

The `scriptVar.save()` function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path `/usb1/`.

If no *filename* is specified (the filename parameter is an empty string), the script is saved to internal nonvolatile memory. Only a script with *filename* defined can be saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

If no *filename* is specified (the filename parameter is an empty string), the script is saved to internal nonvolatile memory. Only a script with *filename* defined can be saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

You can add the file extension, but it is not required. The only allowed extension is `.tsp` (see Example 2).

Example 1

`test8.save()`	Saves the script referenced by the variable `test8` to nonvolatile memory.

Example 2

`test8.save("/usb1/myScript.tsp")`	Saves the script referenced by the variable `test8` to a file named `myScript.tsp` on your flash drive.

Also see

Save a user script (on page 7-10)

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW) (see **Details**)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
scriptVar.source = nil
```

scriptVar	The name of the variable that references the script that contains the source code
code	The body of the script

Details

The `loadscript` or `loadandrunscript` and `endscript` keywords are not included in the source code.

The body of the script is a single string with lines separated by the new line character.

The instrument automatically stores the source for all scripts that are loaded on the instrument. To free up memory or to obfuscate the code, assign `nil` to the source attribute of the script. Although this attribute is writable, it can only be set to the `nil` value.

Example

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
print(test7.source)
```

This example creates a script called "test7" that displays a message on the front panel and retrieves the source code.

Output:

```
display.clear() display.settext('Hello from my test')
```

Also see

scriptVar.list() (on page 8-362)

settime()

This function sets the real-time clock (sets present time of the system).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`settime(time)`

time	The time in seconds since January 1, 1970 UTC

Details

This function sets the date and time of the instrument based on the *time* parameter (specified 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

```
systemTime = os.time({year = 2010,
       month = 3,
       day = 31,
       hour = 14,
       min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2010 at 2:25 pm.

Also see

gettimezone() (on page 8-261)
os.time()
settimezone() (on page 8-367)

settimezone()

This function sets the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
settimezone(offset)
settimezone(offset, dstOffset, dstStart, dstEnd)
```

offset	String representing offset from UTC
dstOffset	String representing the daylight savings offset from UTC
dstStart	String representing when daylight savings time starts
dstEnd	String representing when daylight savings time ends

Details

You only need to set the time zone if you use the `os.time()` and `os.date()` functions.

If only one parameter is given, the same time offset is used throughout the year. If four parameters are given, time is adjusted twice during the year for daylight savings time.

offset and *dstOffset* are strings of the form `"[+|-]hh[:mm[:ss]]"` that indicate how much time must be added to the local time to get UTC time:

- `hh` is a number between 0 and 23 that represents hours
- `mm` is a number between 0 and 59 that represents minutes
- `ss` is a number between 0 and 59 that represents seconds

The minute, second, +, and – fields are optional.

For example, to set the UTC-5 time zone, you specify the string `"5"`, because UTC-5 is 5 hours behind UTC and you must add 5 hours to the local time to determine UTC time. To specify the time zone UTC4, you specify `"-4"`, because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to determine UTC.

dstStart and *dstEnd* are strings of the form `"MM.w.dw/hh[:mm[:ss]]"` that indicate when daylight savings time begins and ends respectively:

- `MM` is a number between 1 and 12 that represents the month
- `w` is a number between 1 and 5 that represents the week in the month
- `dw` is a number between 0 and 6 that represents the day of the week (where 0 is Sunday)

The rest of the fields represent the time of day that the change takes effect:

- `hh` represents hours
- `mm` represents minutes
- `ss` represents seconds

The minutes and seconds fields are optional.

The week of the month and day of the week fields are not specific dates.

Example

`settimezone("8", "1", "3.3.0/02", "11.2.0/02")`	Sets `offset` to equal +8 hours, +1 hour for DST, starts on Mar 14 at 2:00 a.m, ends on Nov 7 at 2:00 a.m.
`settimezone(offset)`	Sets local time zone to `offset`.

Also see

gettimezone() (on page 8-261)
os.time()
settime() (on page 8-366)

setup.cards()

This function returns the card model numbers that are defined for each slot in a saved setup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
CardModels = setup.cards()
CardModels = setup.cards("/path/filename.set")
```

CardModels	A comma-delimited string listing the card model numbers for each slot
path/filename	The path and name of the file on the flash drive; the path may be absolute or relative to the current working directory; the `.set` extension must be on the filename

Details

This function returns a comma-delimited string that lists the card model for each slot in the instrument (from 1 to 6) for the desired saved setup. If no card was installed in the slot when the setup was saved, a 0 is returned as the card model number.

Use *CardModels* = `setup.cards()` to return cards associated with the internally saved setup.

Use *CardModels* = `setup.cards("/path/filename.set")` to return cards associated with the setup saved on the USB flash drive.

Example 1

`CardModels = setup.cards()` `print(CardModels)`	Query the cards associated with the internal saved setup. Output, assuming a Model 3722 card in slot 1: `3722,0,0,0,0,0`
`print(setup.card("/usb1/mysetup.set"))`	Query the cards associated with `mysetup.set` on the root directory on the flash drive. Output, assuming a Model 3723 on slot 2, Model 3722 on slot 3, and Model 3720 on slot 4: `0,3723,3722,3720,0,0`

Example 2

`print(setup.card("/usb1/mysetup.set"))`	Query the cards associated with setup saved as `JulySetup.set` on the thumb drive. The following example of output shows that slots 1, 5, and 6 are empty, slot 2 has a Model 3723 installed, slot 3 has a Model 3722 installed and slot 4 has a model 3720 installed: `0,3723,3722,3720,0,0`

Also see

setup.recall() (on page 8-370)
setup.save() (on page 8-370)

setup.poweron

This attribute specifies which saved setup to recall when the instrument is turned on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
id = setup.poweron
setup.poweron = id
```

id	An integer that specifies the setup to recall when the instrument power is turned on (0 or 1)

Details

When *id* = `0`, the instrument uses the factory default setup when it is turned on. When *id* is set to 1, it uses the setup saved with `setup.save()`.

Only setups stored in nonvolatile memory are available.

To save a script that is used when the instrument is powered on, you can create a configuration script and name it `autoexec`.

Example

`setup.poweron = 0`	Set the instrument to use the factory default setup when power is turned on.

Also see

createconfigscript() (on page 8-115)
Save the present configuration (on page 2-100)
setup.save() (on page 8-370)
Start-up (power-on) configuration (on page 2-35)

setup.recall()

This function recalls settings from a saved setup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setup.recall(id)
```

id	An integer or string that specifies the location of the setup to recall: • Factory default setup (0) • User-saved setup in nonvolatile memory (1) • User-saved setup on a USB flash drive (`"/path/filename"`)

Details

If the *id* parameter is 1, the internal setup that was saved with `setup.save()` is recalled. If the *id* parameter is 0, the instrument recalls the factory default setup.

When the *id* parameter is a string, it is interpreted as the path and file name of the setup to restore from a file on a USB flash drive. The path may be absolute or relative to the current working directory.

Before a setup is recalled, an instrument reset is performed.

Example 1

`setup.recall(1)`	Recall the user-saved setup.

Example 2

`setup.recall("/usb1/KEITHLEY_30730.set")`	Recall a user-saved setup stored in a file named `KEITHLEY_30730` on a USB flash drive.

Also see

Saved setups (on page 2-33)
setup.save() (on page 8-370)

setup.save()

This function saves the present setup as a user-saved setup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
setup.save()
setup.save(id)
```

id	A string that specifies the path and file name to which to save the user setup on a USB flash drive (`"/path/filename"`)

Details

When the *id* parameter is a string, it is interpreted as the path and file name of the location to save the present setup on a USB flash drive. The path may be absolute or relative to the current working directory.

If you do not specify the *id* parameter, the setup is saved to the instrument's nonvolatile memory. If a previous setup exists, it is overwritten.

You can also create configuration scripts to save setups. See Save the present configuration (on page 2-100).

Example

`setup.save()`	Saves the present setup to the internal memory of the instrument.

Also see

createconfigscript() (on page 8-115)
Saved setups (on page 2-33)
setup.recall() (on page 8-370)

slot[X].banks.matrix

This attribute describes the number of banks in the matrix for a card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Never	Not applicable	Not applicable

Usage

```
value = slot[X].banks.matrix
```

value	The number of banks in the matrix
X	The slot number

Details

Returns the number of banks in the matrix on the card in slot *X*. If no matrix or no card exists, it returns nil.

Example

`print(slot[1].banks.matrix)`	Returns the number of banks in the matrix on the card in slot 1 (4 banks). Output: `4.000000000e+000`

Also see

slot[X].columns.matrix (on page 8-372)

slot[X].columns.matrix

This attribute returns the number of columns in the matrix for the card in slot *X*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`value = slot[X].columns.matrix`

value	The number of columns in the matrix
X	The slot number

Details

This attribute is only available for a slot if a card is installed and if the installed card supports matrix channels. If matrix channels are not available, the return value is `nil`.

Example

`print(slot[4].columns.matrix)`	Returns the number of columns in the matrix on the card in slot 4 (28). Example output: `2.800000000e+01`

Also see

slot[X].banks.matrix (on page 8-371)
slot[X].rows.matrix (on page 8-386)

slot[X].commonsideohms

This attribute indicates whether a card in slot *X* supports commonside 4-wire ohm channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`commonside = slot[X].commonsideohms`

commonside	Indication of whether or not commonside 4-wire ohm channels are supported
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports commonside 4-wire ohm channels. If the attribute is not available, the return value is `nil`.

If commonside 4-wire ohms channels are supported, the returned value is `1`.

Example

`print(slot[1].commonsideohms)`	Query if slot 1 supports commonside 4-wire ohms channels.

Also see

None

slot[X].digio

Indicates whether or not a card in slot *X* supports digital I/O channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
value = slot[X].digio
```

value	Indicator for whether or not the card in the slot supports digital I/O channels
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports digital I/O channels. If the attribute is not available, the return value is `nil`.

If digital I/O channels are supported, the returned value is 1.

Example

`print(slot[1].digio)`	Query if slot 1 supports digital I/O channels.

Also see

slot functions and attributes

slot[X].endchannel.*

These attributes indicates whether or not the channel in slot *X* supports a feature and if so, which channels support the feature.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[X].endchannel.amps (channel supports amperage measurements)
value = slot[X].endchannel.analogoutput (channel supports a digital analog output
   (DAC))
value = slot[X].endchannel.digitalio (channel supports digital inputs and outputs)
value = slot[X].endchannel.isolated (channel supports isolated channels)
value = slot[X].endchannel.totalizer (channel supports totalizer channels)
value = slot[X].endchannel.voltage (channel supports voltage or two-wire
   measurements)
```

value	The channel number of the ending channel of the group of channels that supports the feature
X	Slot number (1 to 6.

Details

This attribute is only available for a slot if a card is installed and if the installed card supports the selected feature. If the attribute is not available, the return value is `nil`.

Channels are grouped by feature sets, so you can use the start and ending channel numbers to identify a group of channels that supports a particular feature. If the card supports the feature, the returned value is the number of the ending channel.

If only one channel in the card supports the feature, the ending channel will match the starting channel number.

Example

```
CardChannels = function(SlotNumber)
   if slot[SlotNumber].idn == "Empty Slot" then
      print("   Slot is Empty")
   else
      if (slot[SlotNumber].startchannel.voltage == nil) and
               (slot[SlotNumber].endchannel.voltage == nil) then
         print("   no voltage channels")
      else
         print("   Start voltage channel is " .. slot[SlotNumber].startchannel.voltage)
         print("   End voltage channel is " .. slot[SlotNumber].endchannel.voltage)
      end

      if (slot[SlotNumber].startchannel.amps == nil) and
               (slot[SlotNumber].endchannel.amps == nil) then
         print("   no amp channels")
      else
         print("   Start amp channel is " .. slot[SlotNumber].startchannel.amps)
         print("   End amp channel is " .. slot[SlotNumber].endchannel.amps)
      end

      if (slot[SlotNumber].digio == 1) then
         print("   Start digital i/o channel is " ..
                  slot[SlotNumber].startchannel.digitalio)
         print("   End digital i/o channel is " .. slot[SlotNumber].endchannel.digitalio)
      else
         print("   no digio channels")
      end

      if (slot[SlotNumber].totalizer == 1) then
         print("   Start totalizer channel is " ..
   slot[SlotNumber].startchannel.totalizer)
         print("   End totalizer channel is " .. slot[SlotNumber].endchannel.totalizer)
      else
         print("   no totalizer channels")
      end

      if (slot[SlotNumber].startchannel.analogoutput == nil) and
                (slot[SlotNumber].endchannel.analogoutput == nil) then
         print("   no analog output channels")
      else
         print("   Start analog output channel is " ..
                slot[SlotNumber].startchannel.analogoutput)
         print("   End analog output channel is " ..
                slot[SlotNumber].endchannel.analogoutput)
      end

      if (slot[SlotNumber].matrix == 1) then
         print("   Channels on card are matrix type")
      end
   end
end

for x = 1,6 do
   print("Checking card channels in slot " .. x)
   CardChannels(x)
end
```

If the Series 3700A contains the following cards:

- Slot 1: 3732
- Slot 2: 3720
- Slot 3: 3750
- Slot 4: Empty
- Slot 5: 3721
- Slot 6: Empty

The output of this example is similar to:

```
Checking card channels in slot 1
   no voltage channels
   no amp channels
   no digio channels
   no totalizer channels
   no analog output channels
   Channels on card are matrix type
Checking card channels in slot 2
   Start voltage channel is 1
   End voltage channel is 60
   no amp channels
   no digio channels
   no totalizer channels
   no analog output channels
Checking card channels in slot 3
   no voltage channels
   no amp channels
   Start digital i/o channel is 1
   End digital i/o channel is 5
   Start totalizer channel is 6
   End totalizer channel is 9
   Start analog output channel is 10
   End analog output channel is 11
Checking card channels in slot 4
   Slot is Empty
Checking card channels in slot 5
   Start voltage channel is 1
   End voltage channel is 40
   Start amp channel is 41
   End amp channel is 42
   no digio channels
   no totalizer channels
   no analog output channels
Checking card channels in slot 6
   Slot is Empty
```

Also see

slot functions and attributes
slot[X].startchannel.* (on page 8-386)

slot[X].idn

This attribute returns a string that contains information about the card in slot *X*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

idnString = slot[*X*].idn

idnString	The return string
X	Slot number (1 to 6)

Details

The information that is returned depends on whether the card in the slot is an actual card or pseudocard.

For actual cards, this returns a comma-separated string that contains the model number, description, firmware revision, and serial number of the card installed in slot *X*.

For pseudocards, the response is `Pseudo`, followed by the model number, description, firmware revision, and ??? for the serial number.

Example

`print(slot[3].idn)`	If a Model 3723 is installed in slot 3, the response is: `3723,Dual 1x30 Reed Multiplexer,01.40e,1243657`

Also see

slot[X] attributes (see "Slot" on page 6-18)

slot[X].interlock.override

This attribute suppresses or permits interlock errors to be generated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup	Create configuration script Save setup	0

Usage

```
value = slot[X].interlock.override
slot[X].interlock.override = value
```

value	Indicates the desired state of the interlock override; valid values are `slot.ON` (`1`) or `slot.OFF` (`0`, the default)
X	The slot containing the card to which the interlock state is applied

Details

This command suppresses errors that would otherwise be generated when the interlock is not closed. If the interlock is not physically connected, channels will still not close.

This attribute exists only for installed cards that support detecting an interlock break. Otherwise, the return value is `nil`. If the card supports detecting an interlock break, set this attribute to the desired response.

To enable interlock override on the card, set to `slot.ON`. If an override performed on card is not desired, set to `slot.OFF`. This setting applies to all interlocks on the card.

Example

`slot[3].interlock.override = slot.ON`	Suppresses interlock errors.

Also see

slot[X].interlock.state (on page 8-379)

slot[X].interlock.state

This attribute indicates the interlock state of a card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	1

Usage

`value = slot[X].interlock.state`

value	Indicates whether the interlocks are engaged or not; see table below for possible return values

Details

This attribute will not exist for a slot if a card is not installed or the card installed does not support detecting an interlock break. In these cases, the return value will be `nil`.

Return values for slot*[X]*.interlock.state

Return value	Description
`nil`	No card is installed or the installed card does not support interlocks
`0`	Interlocks 1 and 2 are disengaged on the card
`1`	Interlock 1 is engaged, interlock 2 (if it exists) is disengaged
`2`	Interlock 2 in engaged, interlock 1 is disengaged
`3`	Both interlock 1 and 2 are engaged

Use this attribute to query the interlock state for cards that support detecting interlock break.

Also see

slot[X].interlock.override (on page 8-378)

slot[X].isolated

This attribute indicates if the card in slot *X* supports isolated channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`value = slot[X].isolated`

value	1 if isolated channels are supported
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports the isolated channels. If isolated channels are not available, the return value is `nil`.

Example

`IsolatedChan1 = slot[1].isolated` `print(IsolatedChan1)`	Query if slot 1 supports isolated channels. If it does support isolated channels, the output is: `1.000000000e+00`

Also see

slot[X].idn (on page 8-377)

slot[X].matrix

This attribute indicates if the card in slot *X* supports matrix channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`value = slot[X].matrix`

value	1 if matrix channels are supported
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports matrix channels. If matrix channels are not available, the return value is `nil`.

Example

`Matrix1 = slot[1].matrix` `print(Matrix1)`	Query if slot 1 supports matrix channels. If it does support matrix channels, the output is: `1.000000000e+00`

Also see

slot[X].idn (on page 8-377)

slot[X].maxvoltage

This attribute returns the maximum voltage of all channels on a card in slot *X*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
maximumVolts = slot[X].maxvoltage
```

maximumVolts	The maximum voltage
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports voltage settings. If voltage settings are not available, the return value is `nil`.

Example

`maxVolts2 = slot[2].maxvoltage` `print(maxVolts2)`	Query the maximum voltage on slot 2. The output will be similar to: `3.0000000000e+02`

Also see

slot[X].idn (on page 8-377)

slot[X].multiplexer

This attribute indicates if the card in slot *X* supports multiplexer channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[X].multiplexer
```

value	1 if multiplexer channels are supported
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports multiplexer channels. If multiplexer channels are not available, the return value is `nil`.

Example

`MuxChan1 = slot[1].multiplexer` `print(MuxChan1)`	Query if slot 1 supports multiplexer channels. If it does support multiplexer channels, the output is: `1.000000000e+00`

Also see

slot[X].idn (on page 8-377)

slot[X].poles.four

This attribute indicates if a four-pole setting is supported for the channels on the card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
fourPole = slot[X].poles.four
```

fourPole	The return value
X	Slot number (1 to 6)

Details

This attribute only exists if a card is installed and if the card supports four-pole settings for the channels on the card. If not, the value is nil. If supported, the value is 1.

Example

`fourPole3 = slot[3].poles.four` `print(fourPole3)`	Queries if Slot 3 supports four-pole settings for the channels on the card. Output if card supports four pole: `1.000000000e+00` Output if card does not support four pole: `nil`

Also see

slot[X].poles.one (on page 8-383)
slot[X].poles.two (on page 8-384)

slot[X].poles.one

This attribute indicates if a one-pole setting is supported for the channels on the card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

onePole = slot[*X*].poles.one

onePole	The return value
X	Slot number (1 to 6)

Details

This attribute only exists if a card is installed and if the card supports one-pole settings for the channels on the card. If not, the value is `nil`. If supported, the value is 1.

Example

`print(slot[3].poles.one)`	Query to see if Slot 3 supports one-pole settings for the channels on the card. Output if card supports one pole: `1.000000000e+00` Output if card does not support one pole: `nil`

Also see

slot[X].poles.four (on page 8-382)
slot[X].poles.two (on page 8-384)

slot[X].poles.two

This attribute indicates if a two-pole setting is supported for the channels on the card.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`twoPole = slot[X].poles.two`

twoPole	The return value
X	Slot number (1 to 6)

Details

This attribute only exists is a card is installed and if the card supports a two-pole setting for the channels on the card.

If not, the value is `nil`. If supported, the value is `1`.

Example

`twoPole3 = slot[3].poles.two` `print(twoPole3)`	Query to see if Slot 3 supports two-pole settings for the channels on the card. Output if card supports two pole: `1.000000000e+00` Output if card does not support two pole: `nil`

Also see

slot[X].poles.one (on page 8-383)
slot[X].poles.four (on page 8-382)

slot[X].pseudocard

This attribute specifies the corresponding pseudocard to implement for the designated slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	See **Details**

Usage

```
pseudoCard = slot[X].pseudocard
slot[X].pseudocard = pseudoCard
```

pseudoCard	• Set *pseudocard* to one of the following values: • `slot.PSEUDO_NONE` or `0` for no pseudocard selection • `3720` for Model 3720 Dual 1x30 Multiplexer card simulation • `3721` for Model 3721 Dual 1x20 Multiplex card simulation • `3722` for Model 3722 Dual 1x48 Multiplexer card simulation • `3723` for Model 3723 Dual 1x30 Reed Multiplexer card simulation • `3724` for Model 3724 Dual 1x30 FET Multiplexer card simulation • `3730` for Model 3730 6 x 16 High Density Matrix card simulation • `3731` for Model 3731 6x16 High Speed Reed Relay Matrix card simulation • `3732` or `37320` for Model 3732 Quad 4 x 28 Ultra-High Density Reed Relay Matrix card simulation • `37321` for Model 3732 Dual 4 x 56 Ultra-High Density Reed Relay Matrix card simulation • `37322` for Model 3732 Single 4 x 112 Ultra-High Density Reed Relay Matrix card simulation • `37323` for Model 3732 Dual 8 x 28 Ultra-High Density Reed Relay Matrix card simulation • `37324` for Model 3732 Single 16 x 28 Ultra-High Density Reed Relay Matrix card simulation • `3740` for Model 3740 32-Channel Isolated Switch card simulation • `3750` for Model 3750 Multifunction I/O card
X	Slot number (1 to 6)

Details

This attribute only exists for a slot if that slot has no card installed in it. If a card is installed, the response is `nil` when queried. If no card installed and the slot is empty, the response is 0.

After assigning a pseudocard, the valid commands and attributes based on that pseudocard exist for that slot. For example, the `slot[X].idn` attribute is valid.

Changing the pseudocard card assignment from a card to `slot.PSEUDO_NONE` invalidates existing scan lists that include that slot.

Example

`slot[6].pseudocard = 3720`	Sets the pseudocard of slot 6 for Model 3720 card simulation.

Also see

slot[X] attributes (see "Slot" on page 6-18)
slot[X].idn (on page 8-377)

slot[X].rows.matrix

This attribute returns the number of rows in the matrix on the card in slot X.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
value = slot[X].rows.matrix
```

`value`	Number of rows in the matrix card of the selected slot
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports matrix channels. If matrix channels are not available, the return value is nil.

Example

`print(slot[4].rows.matrix)`	Returns the number of rows in the matrix on the card in slot 4 (12 rows). Example output: `1.2000000000e+01`

Also see

slot[X].columns.matrix (on page 8-372)
slot[X].idn (on page 8-377)
slot[X].matrix (on page 8-380)

slot[X].startchannel.*

These attributes indicates whether or not the channel in slot *X* supports a feature and if so, which channels support the feature.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

`value = slot[X].startchannel.amps` (channel supports amperage measurements)

`value = slot[X].startchannel.analogoutput` (channel supports a digital analog output (DAC))

`value = slot[X].startchannel.digitalio` (channel supports digital inputs and outputs)

`value = slot[X].startchannel.isolated` (channel supports isolated channels)

`value = slot[X].startchannel.totalizer` (channel supports totalizer channels)

`value = slot[X].startchannel.voltage` (channel supports voltage or two-wire measurements)

value	The channel number of the starting channel of the group of channels that supports the feature
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports the selected feature. If the attribute is not available, the return value is `nil`.

Channels are grouped on the cards by feature sets, so you can use the start and ending channel numbers to identify a group of channels that supports a particular feature. If the card supports the feature, the returned value is the number of the starting channel.

If only one channel in the card supports the feature, the starting channel will match the ending channel number.

Example

```
CardChannels = function(SlotNumber)
   if slot[SlotNumber].idn == "Empty Slot" then
      print("   Slot is Empty")
   else
      if (slot[SlotNumber].startchannel.voltage == nil) and
               (slot[SlotNumber].endchannel.voltage == nil) then
         print("   no voltage channels")
      else
         print("   Start voltage channel is " .. slot[SlotNumber].startchannel.voltage)
         print("   End voltage channel is " .. slot[SlotNumber].endchannel.voltage)
      end

      if (slot[SlotNumber].startchannel.amps == nil) and
               (slot[SlotNumber].endchannel.amps == nil) then
         print("   no amp channels")
      else
         print("   Start amp channel is " .. slot[SlotNumber].startchannel.amps)
         print("   End amp channel is " .. slot[SlotNumber].endchannel.amps)
      end

      if (slot[SlotNumber].digio == 1) then
         print("   Start digital i/o channel is " ..
                  slot[SlotNumber].startchannel.digitalio)
         print("   End digital i/o channel is " .. slot[SlotNumber].endchannel.digitalio)
      else
         print("   no digio channels")
      end

      if (slot[SlotNumber].totalizer == 1) then
         print("   Start totalizer channel is " ..
   slot[SlotNumber].startchannel.totalizer)
         print("   End totalizer channel is " .. slot[SlotNumber].endchannel.totalizer)
      else
         print("   no totalizer channels")
      end

      if (slot[SlotNumber].startchannel.analogoutput == nil) and
                (slot[SlotNumber].endchannel.analogoutput == nil) then
         print("   no analog output channels")
      else
         print("   Start analog output channel is " ..
                slot[SlotNumber].startchannel.analogoutput)
         print("   End analog output channel is " ..
                slot[SlotNumber].endchannel.analogoutput)
      end

      if (slot[SlotNumber].matrix == 1) then
         print("   Channels on card are matrix type")
      end
   end
end

for x = 1,6 do
   print("Checking card channels in slot " .. x)
   CardChannels(x)
end
```

If the Series 3700A contains the following cards:

- Slot 1: 3732
- Slot 2: 3720
- Slot 3: 3750
- Slot 4: Empty
- Slot 5: 3721
- Slot 6: Empty

The output of this example is similar to:

```
Checking card channels in slot 1
   no voltage channels
   no amp channels
   no digio channels
   no totalizer channels
   no analog output channels
   Channels on card are matrix type
Checking card channels in slot 2
   Start voltage channel is 1
   End voltage channel is 60
   no amp channels
   no digio channels
   no totalizer channels
   no analog output channels
Checking card channels in slot 3
   no voltage channels
   no amp channels
   Start digital i/o channel is 1
   End digital i/o channel is 5
   Start totalizer channel is 6
   End totalizer channel is 9
   Start analog output channel is 10
   End analog output channel is 11
Checking card channels in slot 4
   Slot is Empty
Checking card channels in slot 5
   Start voltage channel is 1
   End voltage channel is 40
   Start amp channel is 41
   End amp channel is 42
   no digio channels
   no totalizer channels
   no analog output channels
Checking card channels in slot 6
   Slot is Empty
```

Also see

slot functions and attributes
slot[X].endchannel.* (on page 8-373)

slot[X].tempsensor

This attribute indicates if the card in slot *X* 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[X].tempsensor`

value	1 if temperature sensor channels are supported
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card 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: `1.000000000e+00`

Also see

slot[X].idn (on page 8-377)

slot[X].thermal.state

This attribute indicates the thermal state of 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[X].thermal.state`

value	0 if thermal conditions will not affect specifications 1 if thermal conditions are getting warm enough to affect specifications
X	Slot number (1 to 6)

Details

This attribute is only available for a slot if a card is installed and if the installed card supports thermal state detection. If thermal state detection is not available, the return value is `nil`.

Example

`print(slot[3].thermal.state)`	Query the thermal state on slot 3. If spec might be affected by the thermal state, the output is: `1.000000000e+00`

Also see

slot[X].idn (on page 8-377)

status.condition

This attribute stores the status byte condition register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

`statusByte = status.condition`

statusByte	The status byte; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status byte, which is returned as a numeric value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7 **	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred. When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence.

The individual bits of this register have the following meanings:

Bit	Value	Description
B0	`status.MEASUREMENT_SUMMARY_BIT` `status.MSB`	Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	`status.SYSTEM_SUMMARY_BIT` `status.SSB`	Set summary bit indicates that an enabled system event has occurred. Bit B1 decimal value: 2
B2	`status.ERROR_AVAILABLE` `status.EAV`	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	`status.QUESTIONABLE_SUMMARY_BIT` `status.QSB`	Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	`status.MESSAGE_AVAILABLE` `status.MAV`	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	`status.EVENT_SUMMARY_BIT` `status.ESB`	Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32
B6	`status.MASTER_SUMMARY_STATUS` `status.MSS`	Request Service (RQS)/Master Summary Status (MSS). Depending on how it is used, bit B6 of the status byte register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit: • When using the GPIB, USB, or VXI-11 serial poll sequence of the Model 3706A to obtain the status byte (serial poll byte), B6 is the RQS bit. The set bit indicates that the Request Service (RQS) bit of the status byte (serial poll byte) is set and a serial poll (SRQ) has occurred. • When using the `status.condition` register command or the *STB? common command to read the status byte, B6 is the MSS bit. Set bit indicates that an enabled summary bit of the status byte register is set. Bit B6 decimal value: 64
B7	`status.OPERATION_SUMMARY_BIT` `status.OSB`	Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

In addition to the above constants, when 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).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

`statusByte = status.condition` `print(statusByte)`	Returns `statusByte`. Sample output: `1.29000e+02` Converting this output (129) to its binary equivalent yields 1000 0001 Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSS) and B7 (OSB).

Also see

Status byte and service request (SRQ) (on page D-17)

status.measurement.*

This attribute contains the measurement event register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	399 (All bits set)

Usage

```
measurementRegister = status.measurement.condition
measurementRegister = status.measurement.enable
measurementRegister = status.measurement.event
measurementRegister = status.measurement.ntr
measurementRegister = status.measurement.ptr
status.measurement.enable = measurementRegister
status.measurement.ntr = measurementRegister
status.measurement.ptr = measurementRegister
```

measurementRegister	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits);the only valid value other than 0 is 8

Details

These attributes read or write the measurement event registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume value 384 is returned for the enable register. The binary equivalent is 0000 0001 1000 0000. This value indicates that bit B7 (ROF) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.measurement.LOWER_LIMIT1` `status.measurement.LLMT1`	Set bit indicates that a reading has exceeded the lower limit 1 value. Bit B0 decimal value: 1
B1	`status.measurement.UPPER_LIMIT1` `status.measurement.ULMT1`	Set bit indicates that a reading has exceeded the upper limit 1 value. Bit B1 decimal value: 2
B2	`status.measurement.LOWER_LIMIT2` `status.measurement.LLMT2`	Set bit indicates that a reading has exceeded the lower limit 2 value. Bit B2 decimal value: 4
B3	`status.measurement.UPPER_LIMIT2` `status.measurement.ULMT2`	Set bit indicates that a reading has exceeded the upper limit 2 value. Bit B3 decimal value: 8
B4-B6	Not used	Not applicable
B7	`status.measurement.READING_OVERFLOW` `status.measurement.ROF`	Set bit indicates that a reading has resulted in an overflow measurement value. Bit B7 decimal value: 128
B8	`status.measurement.BUFFER_AVAILABLE` `status.measurement.BAV`	Set bit indicates that a reading buffer is storing measurement values. Bit B8 decimal value: 256 Binary value: 0001 0000 0000
B9-B15	Not used	Not applicable

For example, assume value 257 is returned for the enable register. The binary equivalent is 0000 0001 0000 0001. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents and Enable and transition registers. The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.measurement.VOLTAGE_LIMIT` `status.measurement.VLMT`	Set bit is a summary of the `status.measurement.voltage_limit` register. Bit B0 decimal value: 1
B1	`status.measurement.CURRENT_LIMIT` `status.measurement.ILMT`	Set bit is a summary of the `status.measurement.current_limit` register. Bit B1 decimal value: 2
B2-B6	Not used	Not applicable
B7	`status.measurement.READING_OVERFLOW` `status.measurement.ROF`	Set bit is a summary of the `status.measurement.reading_overflow` register. Bit B7 decimal value: 128
B8	`status.measurement.BUFFER_AVAILABLE` `status.measurement.BAV`	Set bit is a summary of the `status.measurement.buffer_available` register. Bit B8 decimal value: 256
B9-B10	Not used	Not applicable
B11	`status.measurement.OUTPUT_ENABLE` `status.measurement.OE`	Model 2601A/2602A/2604A: output enable line. Set bit indicates that output enable has been asserted. Bit B11 decimal value: 2,048
	`status.measurement.INTERLOCK` `status.measurement.INT`	Model 2611A/2612A/2614A/2635A/2636A: interlock line. Set bit indicates that interlock has been asserted. Bit B11 decimal value: 2,048
B12	Not used	Not applicable
B13	`status.measurement.INSTRUMENT_SUMMARY` `status.measurement.INST`	Set bit indicates that a bit in the measurement instrument summary register is set. Bit B13 decimal value: 8,192
B14-B15	Not used	Not applicable

As an example, to set bit B8 of the measurement event enable register, set `status.measurement.enable = status.measurement.BAV`.

In addition to the above constants, *`measurementRegister`* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *`measurementRegister`* to the sum of their decimal weights. For example, to set bits B1 and B8, set *`measurementRegister`* to 258 (which is the sum of 2 + 256).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

`status.measurement.enable = status.measurement.BAV`	Sets the BAV bit of the measurement event enable register.

Also see

Measurement event registers

status.node_enable

This attribute stores the system node enable register. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
nodeEnableRegister = status.node_enable
status.node_enable = nodeEnableRegister
```

nodeEnableRegister	The status of the system node enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

This attribute is used to read or write to the system node enable register. Reading the system node enable register returns a value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, assume the value of `1.29000e+02` (which is 129) is returned for the system node enable register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, a summary bit is set in the appropriate system summary register. The register and bit that is set depends on the TSP-Link node number assigned to this instrument.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	**Value**	**Description**
B0	`status.MEASUREMENT_SUMMARY_BIT` `status.MSB`	Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	Not used	Not applicable.
B2	`status.ERROR_AVAILABLE` `status.EAV`	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	`status.QUESTIONABLE_SUMMARY_BIT` `status.QSB`	Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	`status.MESSAGE_AVAILABLE` `status.MAV`	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	`status.EVENT_SUMMARY_BIT` `status.ESB`	Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32
B6	`status.MASTER_SUMMARY_STATUS` `status.MSS`	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set. Bit B6 decimal value: 64
B7	`status.OPERATION_SUMMARY_BIT` `status.OSB`	Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

As an example, to set the B0 bit of the system node enable register, set `status.node_enable = status.MSB`.

In addition to the above values, *`nodeEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`nodeEnableRegister`* to the sum of their decimal weights. For example, to set bits B0 and B7, set *`nodeEnableRegister`* to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
nodeEnableRegister = status.MSB + status.OSB
status.node_enable = nodeEnableRegister
```

Sets the MSB and OSB bits of the system node enable register using constants.

Example 2

```
-- decimal 129 = binary 10000001
nodeEnableRegister = 129
status.node_enable = nodeEnableRegister
```

Sets the MSB and OSB bits of the system node enable register using a decimal value.

Also see

status.condition (on page 8-391)
status.system.* (on page 8-411)
Status byte and service request (SRQ) (on page D-17)

status.node_event

This attribute stores the status node event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

`nodeEventRegister = status.node_event`

nodeEventRegister	The status of the node event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the status node event register, which is returned as a numeric value (reading this register returns a value). The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7 **	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred.

Bit	Value	Description
B0	`status.MEASUREMENT_SUMMARY_BIT` `status.MSB`	Set summary bit indicates that an enabled measurement event has occurred. Bit B0 decimal value: 1
B1	Not used	Not applicable
B2	`status.ERROR_AVAILABLE` `status.EAV`	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	`status.QUESTIONABLE_SUMMARY_BIT` `status.QSB`	Set summary bit indicates that an enabled questionable event has occurred. Bit B3 decimal value: 8
B4	`status.MESSAGE_AVAILABLE` `status.MAV`	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	`status.EVENT_SUMMARY_BIT` `status.ESB`	Set summary bit indicates that an enabled standard event has occurred. Bit B5 decimal value: 32
B6	`status.MASTER_SUMMARY_STATUS` `status.MSS`	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte register is set. Bit B6 decimal value: 64
B7	`status.OPERATION_SUMMARY_BIT` `status.OSB`	Set summary bit indicates that an enabled operation event has occurred. Bit B7 decimal value: 128

In addition to the above constants, *nodeEventRegister* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *nodeEventRegister* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

`nodeEventRegister = status.node_event` `print(nodeEventRegister)`	Reads the status node event register. Sample output: `1.29000e+02` Converting this output (129) to its binary equivalent yields 1000 0001 Therefore, this output indicates that the set bits of the status byte condition register are presently B0 (MSB) and B7 (OSB).

Also see

Status byte and service request (SRQ) (on page D-17)
status.condition (on page 8-391)
status.system.* (on page 8-411)

status.operation.*

These attributes manage the operation status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	22545

Usage

```
operationRegister = status.operation.condition
operationRegister = status.operation.enable
operationRegister = status.operation.event
operationRegister = status.operation.ntr
operationRegister = status.operation.ptr
status.operation.enable = operationRegister
status.operation.ntr = operationRegister
status.operation.ptr = operationRegister
```

operationRegister	The status of the operation status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes read or write the operation status registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `2.04800e+04` (which is 20,480) is read as the value of the condition register, the binary equivalent is 0101 0000 0000 0000. This value indicates that bit B14 (`PROGRAM_RUNNING`) and bit B12 (USER) are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.operation.CALIBRATING` `status.operation.CAL`	Set bit indicates that the DMM is calibrating. Bit B0 decimal value: 1
B1-B3	`Not used`	Not applicable
B4	`status.operation.MEASURING` `status.operation.MEAS`	Set bit indicates that DMM is measuring. Bit B4 decimal value: 16
B5-B9	`Not used`	Not applicable
B11	`status.operation.PROMPTS` `status.operation.PRMPTS`	Set bit indicates that the command prompts are enabled. Bit B11 decimal value: 2,048
B12	`status.operation.USER`	Set bit indicates that the summary bit from the `status.operation.user` register is set. Bit B12 decimal value: 4,096
B14	`status.operation.PROGRAM_RUNNING` `status.operation.PROG`	Set bit indicates that a command or program is running. Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B12 of the operation status enable register, set `status.operation.enable = status.operation.USER`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B12 and B14, set *`operationRegister`* to 20,480 (which is the sum of 4,096 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

The used bits of the operation event registers are:

- Bit B0, CAL - Set bit indicates that the instrument is calibrating.
- Bit B4, MEAS - Bit is set when taking a measurement.
- Bit B11, PRMPTS - Set bit indicates that command prompts are enabled.
- Bit B12, USER - Set bit indicates that an enabled bit in the operation status user register is set.
- Bit B14, PROG - Set bit indicates that a program is running.

Example 1

```
operationRegister = status.operation.USER +
   status.operation.PROG
status.operation.enable = operationRegister
```

Sets the USER and PROG bits of the operation status enable register using constants.

Example 2

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Sets the USER and PROG bits of the operation status enable register using a decimal value.

Also see

Operation Status Registers

status.operation.user.*

These attributes manage the operation status user register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (RW)	Yes	Status reset	Not saved	0
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
operationRegister = status.operation.user.condition
operationRegister = status.operation.user.enable
operationRegister = status.operation.user.event
operationRegister = status.operation.user.ntr
operationRegister = status.operation.user.ptr
status.operation.user.condition = operationRegister
status.operation.user.enable = operationRegister
status.operation.user.ntr = operationRegister
status.operation.user.ptr = operationRegister
```

operationRegister	The status of the operation status user register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status user registers. Reading a status register returns a value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bits B0 and B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.operation.user.BIT0`	Bit B0 decimal value: 1
B1	`status.operation.user.BIT1`	Bit B1 decimal value: 2
B2	`status.operation.user.BIT2`	Bit B2 decimal value: 4
B3	`status.operation.user.BIT3`	Bit B3 decimal value: 8
B4	`status.operation.user.BIT4`	Bit B4 decimal value: 16
B5	`status.operation.user.BIT5`	Bit B5 decimal value: 32
B6	`status.operation.user.BIT6`	Bit B6 decimal value: 64
B7	`status.operation.user.BIT7`	Bit B7 decimal value: 128
B8	`status.operation.user.BIT8`	Bit B8 decimal value: 256
B9	`status.operation.user.BIT9`	Bit B9 decimal value: 512
B10	`status.operation.user.BIT10`	Bit B10 decimal value: 1,024
B11	`status.operation.user.BIT11`	Bit B11 decimal value: 2,048
B12	`status.operation.user.BIT12`	Bit B12 decimal value: 4,096
B13	`status.operation.user.BIT13`	Bit B13 decimal value: 8,192
B14	`status.operation.user.BIT14`	Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the operation status user enable register, set `status.operation.user.enable = status.operation.user.BIT0`.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B11 and B14, set *operationRegister* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.user.BIT11 +
   status.operation.user.BIT14
status.operation.user.enable = operationRegister
```

Uses constants to set bits B11 and B14 of the operation status user enable register.

Example 2

```
-- 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Uses a decimal value to set bits B11 and B14 of the operation status user enable register.

Also see

Operation Status Register
status.operation.* (on page 8-399)

status.questionable.*

These attributes manage the status model's questionable status register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,256 (All bits set)

Usage

```
questionableRegister = status.questionable.condition
questionableRegister = status.questionable.enable
questionableRegister = status.questionable.event
questionableRegister = status.questionable.ntr
questionableRegister = status.questionable.ptr
status.questionable.enable = questionableRegister
status.questionable.ntr = questionableRegister
status.questionable.ptr = questionableRegister
```

questionableRegister	The status of the questionable status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the questionable status registers. Reading a status register returns a value. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of 1.22880e+04 (which is 12,288) is read as the value of 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
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not available
B1	`status.questionable.SLOT1_INTERLOCK` `status.questionable.S1INL`	Sets the interlock connection of the card in slot 1. Bit B1 decimal value: 2
B2	`status.questionable.SLOT2_INTERLOCK` `status.questionable.S2INL`	Sets the interlock connection of the card in slot 2. Bit B2 decimal value: 4
B3	`status.questionable.SLOT3_INTERLOCK` `status.questionable.S3INL`	Sets the interlock connection of the card in slot 3. Bit B3 decimal value: 8
B4	`status.questionable.SLOT4_INTERLOCK` `status.questionable.S4INL`	Sets the interlock connection of the card in slot 4. Bit B4 decimal value: 16
B5	`status.questionable.SLOT5_INTERLOCK` `status.questionable.S5INL`	Sets the interlock connection of the card in slot 5. Bit B5 decimal value: 32
B6	`status.questionable.SLOT6_INTERLOCK` `status.questionable.S6INL`	Sets the interlock connection of the card in slot 6. Bit B6 decimal value: 64
B7	`status.questionable.DMM_CONNECTION` `status.questionable.DMMCONN`	Indicates that the DMM connection in in question for a measurement taken. Bit B7 decimal value: 128
B8	`status.questionable.CALIBRATION` `status.questionable.CAL`	Indicates that the calibration of the instrument is in question. Bit B8 decimal value: 256
B9	`status.questionable.S1THR` `status.questionable.SLOT1_THERMAL`	Indicates that the thermal functions of the card in slot 1 are questionable. Bit B9 decimal value: 512
B10	`status.questionable.S2THR` `status.questionable.SLOT2_THERMAL`	Indicates that the thermal functions of the card in slot 2 are questionable. Bit B10 decimal value: 1,024
B11	`status.questionable.S3THR` `status.questionable.SLOT3_THERMAL`	Indicates that the thermal functions of the card in slot 3 are questionable. Bit B11 decimal value: 2,048
B12	`status.questionable.S4THR` `status.questionable.SLOT4_THERMAL`	Indicates that the thermal functions of the card in slot 4 are questionable. Bit B12 decimal value: 4,096
B13	`status.questionable.S5THR` `status.questionable.SLOT5_THERMAL`	Indicates that the thermal functions of the card in slot 5 are questionable. Bit B13 decimal value: 8,192
B14	`status.questionable.S6THR` `status.questionable.SLOT6_THERMAL`	Indicates that the thermal functions of the card in slot 6 are questionable. Bit B14 decimal value: 16,384
B15	Not used	Not available

As an example, to set bit B9 of the questionable status enable register, set `status.questionable.enable = status.questionable.SLOT1_THERMAL`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights. For example, to set bits B12 and B13, set *`questionableRegister`* to 12,288 (which is the sum of 4,096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
questionableRegister = status.questionable.S1INL
    +
     status.questionable.S6INL
status.questionable.enable = questionableRegister
```

Uses constants to set bits B1 and B6 of the status questionable enable register.

Example 2

```
-- 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.

Also see

Questionable Status Registers

status.request_enable

This attribute stores the service request (SRQ) enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
requestSRQEnableRegister = status.request_enable
status.request_enable = requestSRQEnableRegister
```

`requestSRQEnableRegister`	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

This attribute is used to read or write to the service request enable register. Reading the service request enable register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	**Value**	**Description**
B0	`status.MEASUREMENT_SUMMARY_BIT` `status.MSB`	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred. Bit B0 decimal value: 1
B1	`status.SYSTEM_SUMMARY_BIT` `status.SSB`	This bit is only available on Models 2601A/2602A/2611A/2612A/2635A/2636A. Set summary bit indicates that an enabled event in the System Summary Register has occurred. Bit B1 decimal value: 2
B2	`status.ERROR_AVAILABLE` `status.EAV`	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	`status.QUESTIONABLE_SUMMARY_BIT` `status.QSB`	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred. Bit B3 decimal value: 8
B4	`status.MESSAGE_AVAILABLE` `status.MAV`	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	`status.EVENT_SUMMARY_BIT` `status.ESB`	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred. Bit B5 decimal value: 32
B6	Not used	Not applicable
B7	`status.OPERATION_SUMMARY_BIT` `status.OSB`	Set summary bit indicates that an enabled event in the Operation Status Register has occurred. Bit B7 decimal value: 128

As an example, to set bit B0 of the service request enable register, set `status.request_enable = status.MSB`.

In addition to the above values, *`requestSRQEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`requestSRQEnableRegister`* to the sum of their decimal weights. For example, to set bits B0 and B7, set *`requestSRQEnableRegister`* to 129 (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

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.

Example 2

`-- decimal 129 = binary 10000001` `requestSRQEnableRegister = 129` `status.request_enable = requestSRQEnableRegister`	Uses a decimal value to set the MSB and OSB bits of the service request (SRQ) enable register.

Also see

Status byte and service request (SRQ) (on page D-17)
status.condition (on page 8-391)
status.system.* (on page 8-411)

status.request_event

This attribute stores the service request (SRQ) event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

`requestSRQEventRegister` `= status.request_event`

`requestSRQEventRegister`	The status of the request event register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

This attribute is used to read the service request event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.29000e+02` (which is 129) is read as the value of this register, the binary equivalent is 1000 0001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	**Value**	**Description**
B0	`status.MEASUREMENT_SUMMARY_BIT` `status.MSB`	Set summary bit indicates that an enabled event in the Measurement Event Register has occurred. Bit B0 decimal value: 1
B1	`status.SYSTEM_SUMMARY_BIT` `status.SSB`	This bit is only available on Models 2601A/2602A/2611A/2612A/2635A/2636A. Set summary bit indicates that an enabled event in the System Summary Register has occurred. Bit B1 decimal value: 2
B2	`status.ERROR_AVAILABLE` `status.EAV`	Set summary bit indicates that an error or status message is present in the Error Queue. Bit B2 decimal value: 4
B3	`status.QUESTIONABLE_SUMMARY_BIT` `status.QSB`	Set summary bit indicates that an enabled event in the Questionable Status Register has occurred. Bit B3 decimal value: 8
B4	`status.MESSAGE_AVAILABLE` `status.MAV`	Set summary bit indicates that a response message is present in the Output Queue. Bit B4 decimal value: 16
B5	`status.EVENT_SUMMARY_BIT` `status.ESB`	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred. Bit B5 decimal value: 32
B6	Not used	Not applicable
B7	`status.OPERATION_SUMMARY_BIT` `status.OSB`	Set summary bit indicates that an enabled event in the Operation Status Register has occurred. Bit B7 decimal value: 128

In addition to the above constants, *`requestEventRegister`* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *`requestEventRegister`* contains the sum of their decimal weights. For example, if 129 is returned, bits B0 and B7 are set (1 + 128).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

`requestEventRegister = status.request_event` `print(requestEventRegister)`	Reads the status request event register. Sample output: `1.29000e+02` Converting this output (129) to its binary equivalent yields 1000 0001. Therefore, this output indicates that the set bits of the status request event register are presently B0 (MSB) and B7 (OSB).

Also see

status.condition (on page 8-391)
status.system.* (on page 8-411)
Status byte and service request (SRQ) (on page D-17)

status.reset()

This function resets all bits in the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.reset()
```

Details

This function clears all status data structure registers (enable, event, NTR, and PTR) to their default values. For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21).

Example

`status.reset()`	Resets the instrument status model.

Also see

Status model (on page 6-18, on page D-1, "*Status Byte Register overview*" on page D-4)

status.standard.*

These attributes manage the standard event status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	253 (All bits set)

Usage

```
standardRegister = status.standard.condition
standardRegister = status.standard.enable
standardRegister = status.standard.event
standardRegister = status.standard.ntr
standardRegister = status.standard.ptr
status.standard.enable = standardRegister
status.standard.ntr = standardRegister
status.standard.ptr = standardRegister
```

standardRegister	The status of the standard event status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the standard event status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.standard.OPERATION_COMPLETE` `status.standard.OPC`	Set bit indicates that all pending selected instrument operations are completed and the instrument is ready to accept new commands. The bit is set in response to an `*OPC` command. The `opc()` function can be used in place of the `*OPC` command. Bit B0 decimal value: 1
B1	Not used	Not applicable
B2	`status.standard.QUERY_ERROR` `status.standard.QYE`	Set bit indicates that you attempted to read data from an empty Output Queue. Bit B2 decimal value: 4
B3	`status.standard.DEVICE_DEPENDENT_ERROR` `status.standard.DDE`	Set bit indicates that an instrument operation did not execute properly due to some internal condition. Bit B3 decimal value: 8
B4	`status.standard.EXECUTION_ERROR` `status.standard.EXE`	Set bit indicates that the instrument detected an error while trying to execute a command. Bit B4 decimal value: 16
B5	`status.standard.COMMAND_ERROR` `status.standard.CME`	Set bit indicates that a command error has occurred. Command errors include: **IEEE Std 488.2 syntax error:** Instrument received a message that does not follow the defined syntax of the IEEE Std 488.2 standard. **Semantic error:** Instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented. **GET error:** The instrument received a Group Execute Trigger (GET) inside a program message. Bit B5 decimal value: 32
B6	`status.standard.USER_REQUEST` `status.standard.URQ`	Set bit indicates that the LOCAL key on the instrument front panel was pressed. Bit B6 decimal value: 64
B7	`status.standard.POWER_ON` `status.standard.PON`	Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read. Bit B7 decimal value: 128
B8-B15	Not used	Not applicable

As an example, to set bit B0 of the standard event status enable register, set `status.standard.enable = status.standard.OPC`.

In addition to the above constants, *`standardRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`standardRegister`* to the sum of their decimal weights. For example, to set bits B0 and B4, set *`standardRegister`* to 17 (which is the sum of 1 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
standardRegister = status.standard.OPC
    + status.standard.EXE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and EXE bits of the standard event status enable register.

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Uses a decimal value to set the OPC and EXE bits of the standard event status enable register.

Also see

Event summary bit (ESB register) (on page D-11)

status.system.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 1 through 14. These attributes are not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system.condition
enableRegister = status.system.enable
enableRegister = status.system.event
enableRegister = status.system.ntr
enableRegister = status.system.ptr
status.system.enable = enableRegister
status.system.ntr = enableRegister
status.system.ptr = enableRegister
```

`enableRegister`	The status of the system summary register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	**Value**	**Description**
B0	`status.system.EXTENSION_BIT` `status.system.EXT`	Bit B0 decimal value: 1
B1	`status.system.NODE1`	Bit B1 decimal value: 2
B2	`status.system.NODE2`	Bit B2 decimal value: 4
B3	`status.system.NODE3`	Bit B3 decimal value: 8
B4	`status.system.NODE4`	Bit B4 decimal value: 16
B5	`status.system.NODE5`	Bit B5 decimal value: 32
B6	`status.system.NODE6`	Bit B6 decimal value: 64
B7	`status.system.NODE7`	Bit B7 decimal value: 128
B8	`status.system.NODE8`	Bit B8 decimal value: 256
B9	`status.system.NODE9`	Bit B9 decimal value: 512
B10	`status.system.NODE10`	Bit B10 decimal value: 1,024
B11	`status.system.NODE11`	Bit B11 decimal value: 2,048
B12	`status.system.NODE12`	Bit B12 decimal value: 4,096
B13	`status.system.NODE13`	Bit B13 decimal value: 8,192
B14	`status.system.NODE14`	Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary status enable register, set `status.system.enable = status.system.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

`enableRegister = status.system.NODE11 +` `    status.system.NODE14` `status.system.enable = enableRegister`	Uses constants to set bits B11 and B14 of the system summary enable register.

Example 2

`-- decimal 18432 = binary 0100 1000 0000 0000` `enableRegister = 18432` `status.system.enable = enableRegister`	Uses a decimal value to set bits B11 and B14 of the system summary enable register.

Also see

status.system2.* (on page 8-413)
System summary and standard event registers (see "System summary bit (System register)" on page D-5)

status.system2.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 15 through 28. These attributes are not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system2.condition
enableRegister = status.system2.enable
enableRegister = status.system2.event
enableRegister = status.system2.ntr
enableRegister = status.system2.ptr
status.system2.enable = enableRegister
status.system2.ntr = enableRegister
status.system2.ptr = enableRegister
```

enableRegister	The status of the system summary 2 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	**Value**	**Description**
B0	`status.system2.EXTENSION_BIT` `status.system2.EXT`	Bit B0 decimal value: 1
B1	`status.system2.NODE15`	Bit B1 decimal value: 2
B2	`status.system2.NODE16`	Bit B2 decimal value: 4
B3	`status.system2.NODE17`	Bit B3 decimal value: 8
B4	`status.system2.NODE18`	Bit B4 decimal value: 16
B5	`status.system2.NODE19`	Bit B5 decimal value: 32
B6	`status.system2.NODE20`	Bit B6 decimal value: 64
B7	`status.system2.NODE21`	Bit B7 decimal value: 128
B8	`status.system2.NODE22`	Bit B8 decimal value: 256
B9	`status.system2.NODE23`	Bit B9 decimal value: 512
B10	`status.system2.NODE24`	Bit B10 decimal value: 1,024
B11	`status.system2.NODE25`	Bit B11 decimal value: 2,048
B12	`status.system2.NODE26`	Bit B12 decimal value: 4,096
B13	`status.system2.NODE27`	Bit B13 decimal value: 8,192
B14	`status.system2.NODE28`	Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 2 enable register, set `status.system2.enable = status.system2.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system2.NODE25 +
    status.system2.NODE28
status.system2.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 2 enable register.

Example 2

`-- decimal 18432 = binary 0100 1000 0000 0000` `enableRegister = 18432` `status.system2.enable = enableRegister`	Uses a decimal value to set bits B11 and B14 of the system summary 2 enable register.

Also see

status.system.* (on page 8-411)
status.system3.* (on page 8-415)
System summary and standard event registers (see "System summary bit (System register)" on page D-5)

status.system3.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 29 through 42. These attributes are not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system3.condition
enableRegister = status.system3.enable
enableRegister = status.system3.event
enableRegister = status.system3.ntr
enableRegister = status.system3.ptr
status.system3.enable = enableRegister
status.system3.ntr = enableRegister
status.system3.ptr = enableRegister
```

enableRegister	The status of the system summary 3 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.system3.EXTENSION_BIT` `status.system3.EXT`	Bit B0 decimal value: 1
B1	`status.system3.NODE29`	Bit B1 decimal value: 2
B2	`status.system3.NODE30`	Bit B2 decimal value: 4
B3	`status.system3.NODE31`	Bit B3 decimal value: 8
B4	`status.system3.NODE32`	Bit B4 decimal value: 16
B5	`status.system3.NODE33`	Bit B5 decimal value: 32
B6	`status.system3.NODE34`	Bit B6 decimal value: 64
B7	`status.system3.NODE35`	Bit B7 decimal value: 128
B8	`status.system3.NODE36`	Bit B8 decimal value: 256
B9	`status.system3.NODE37`	Bit B9 decimal value: 512
B10	`status.system3.NODE38`	Bit B10 decimal value: 1,024
B11	`status.system3.NODE39`	Bit B11 decimal value: 2,048
B12	`status.system3.NODE40`	Bit B12 decimal value: 4,096
B13	`status.system3.NODE41`	Bit B13 decimal value: 8,192
B14	`status.system3.NODE42`	Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 3 enable register, set `status.system3.enable = status.system3.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system3.NODE39 +
    status.system3.NODE42
status.system3.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 3 enable register.

Example 2

`-- decimal 18432 = binary 0100 1000 0000 0000` `enableRegister = 18432` `status.system3.enable = enableRegister`	Uses a decimal value to set bits B11 and B14 of the system summary 3 enable register.

Also see

status.system2.* (on page 8-413)
status.system4.* (on page 8-417)
System summary and standard event registers (see "System summary bit (System register)" on page D-5)

status.system4.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 43 through 56. These attributes are not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (All bits set)

Usage

```
enableRegister = status.system4.condition
enableRegister = status.system4.enable
enableRegister = status.system4.event
enableRegister = status.system4.ntr
enableRegister = status.system4.ptr
status.system4.enable = enableRegister
status.system4.ntr = enableRegister
status.system4.ptr = enableRegister
```

enableRegister	The status of the system summary 4 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	`status.system4.EXTENSION_BIT` `status.system4.EXT`	Bit B0 decimal value: 1
B1	`status.system4.NODE43`	Bit B1 decimal value: 2
B2	`status.system4.NODE44`	Bit B2 decimal value: 4
B3	`status.system4.NODE45`	Bit B3 decimal value: 8
B4	`status.system4.NODE46`	Bit B4 decimal value: 16
B5	`status.system4.NODE47`	Bit B5 decimal value: 32
B6	`status.system4.NODE48`	Bit B6 decimal value: 64
B7	`status.system4.NODE49`	Bit B7 decimal value: 128
B8	`status.system4.NODE50`	Bit B8 decimal value: 256
B9	`status.system4.NODE51`	Bit B9 decimal value: 512
B10	`status.system4.NODE52`	Bit B10 decimal value: 1,024
B11	`status.system4.NODE53`	Bit B11 decimal value: 2,048
B12	`status.system4.NODE54`	Bit B12 decimal value: 4,096
B13	`status.system4.NODE55`	Bit B13 decimal value: 8,192
B14	`status.system4.NODE56`	Bit B14 decimal value: 16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 4 enable register, set `status.system4.enable = status.system4.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B11 and B14, set *`enableRegister`* to 18,432 (which is the sum of 2,048 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system4.NODE53 +
    status.system4.NODE56
status.system2.enable = enableRegister
```

Uses constants to set bit B11 and bit B14 of the system summary 4 enable register.

Example 2

`-- decimal 18432 = binary 0100 1000 0000 0000` `enableRegister = 18432` `status.system4.enable = enableRegister`	Uses a decimal value to set bit B11 and bit B14 of the system summary 4 enable register.

Also see

status.system3.* (on page 8-415)
status.system5.* (on page 8-419)
System summary and standard event registers (see "System summary bit (System register)" on page D-5)

status.system5.*

These attributes manage the TSP-Link® system summary register of the status model for nodes 57 through 64. These attributes are not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	- -	- -	- -	- -
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	510 (All bits set)

Usage

```
enableRegister = status.system5.condition
enableRegister = status.system5.enable
enableRegister = status.system5.event
enableRegister = status.system5.ntr
enableRegister = status.system5.ptr
status.system5.enable = enableRegister
status.system5.ntr = enableRegister
status.system5.ptr = enableRegister
```

enableRegister	The status of the system summary 5 register; a zero (0) indicates no bits set; other values indicate various bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.30000e+02` (which is 130) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0010. This value indicates that bit B1 and bit B7 are set.

B15 **	B14 >	B13 >	B12 >	B11 >	B10 >	B9 >	B8 >	B7 >	B6 >	B5 >	B4 >	B3 >	B2 >	B1 >	B0 *
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page D-2) and Enable and transition registers (on page D-21). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable
B1	`status.system5.NODE57`	Bit B1 decimal value: 2
B2	`status.system5.NODE58`	Bit B2 decimal value: 4
B3	`status.system5.NODE59`	Bit B3 decimal value: 8
B4	`status.system5.NODE60`	Bit B4 decimal value: 16
B5	`status.system5.NODE61`	Bit B5 decimal value: 32
B6	`status.system5.NODE62`	Bit B6 decimal value: 64
B7	`status.system5.NODE63`	Bit B7 decimal value: 128
B8	`status.system5.NODE64`	Bit B8 decimal value: 256
B9-B15	Not used	Not applicable

As an example, to set bit B1 of the system summary 5 enable register, set `status.system5.enable = status.system5.NODE57`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`enableRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Bit	B15	B14	B13	B12	B11	B10	B9	B8
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system5.NODE57 +
    status.system5.NODE60
status.system2.enable = enableRegister
```

Uses constants to set bits B1 and B4 of the system summary 5 enable register.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Uses a decimal value to set bits B1 and B4 of the system summary 5 enable register.

Also see

status.system4.* (on page 8-417)
System summary and standard event registers (see "System summary bit (System register)" on page D-5)

timer.measure.t()

This function measures the elapsed time since the timer was last reset.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
time = timer.measure.t()
```

time	The elapsed time in seconds (1 µs resolution)

Example 1

`timer.reset()` `-- (intervening code)` `time = timer.measure.t()` `print(time)`	Resets the timer and measures the time since the reset. Output: `1.469077e+01` The output will vary. The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Example 2

`beeper.beep(0.5, 2400)` `print("reset timer")` `timer.reset()` `delay(0.5)` `dt = timer.measure.t()` `print("timer after delay:", dt)` `beeper.beep(0.5, 2400)`	Sets the beeper, resets the timer, sets a delay, then verifies the time of the delay before the next beeper. Output: `reset timer` `timer after delay: 5.00e-01`

Also see

timer.reset() (on page 8-421)

timer.reset()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timer.reset()
```

Example

`timer.reset()` `-- (intervening code)` `time = timer.measure.t()` `print(time)`	Resets the timer and then measures the time since the reset. Output: `1.469077e+01` The above output indicates that `timer.measure.t()` was executed 14.69077 seconds after `timer.reset()`.

Also see

timer.measure.t() (on page 8-421)

trigger.blender[N].clear()

This function clears the blender event detector and resets blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

N	The blender number (1 or 2)

Details

This command sets the blender event detector to the undetected state and resets the event detector's overrun indicator.

Example

`trigger.blender[2].clear()`	Clears the event detector for blender 2.

Also see

None

trigger.blender[N].EVENT_ID

This constant contains the trigger blender event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.blender[N].EVENT_ID
```

eventID	Trigger event number
N	The blender number (1 or 2)

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this trigger blender.

Example

`digio.trigger[1].stimulus = trigger.blender[2].EVENT_ID`	Set the trigger stimulus of digital I/O trigger 1 to be controlled by the trigger blender 2 event.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender operates in OR mode or AND mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Trigger blender N reset Recall setup	Create configuration script Save setup	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The orenable mode: • `true`: OR mode • `false`: AND mode
N	The trigger blender (1 or 2)

Details

This command selects whether the blender waits for any one event (the “OR” mode) or waits for all selected events (the “AND” mode) before signaling an output event.

Example

`trigger.blender[1].orenable = true` `trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID` `trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID`	Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 8-424)

trigger.blender[N].overrun

This attribute indicates whether or not an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Trigger blender N clear Trigger blender N reset	Not applicable	Not applicable

Usage

`overrun = trigger.blender[N].overrun`

overrun	Trigger blender overrun state (`true` or `false`)
N	The blender number (1 or 2)

Details

Indicates if an event was ignored because the event detector was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

`print(trigger.blender[1].overrun)`	If an event was ignored, the output is `true`. If an event was not ignored, the output is `false`.

Also see

trigger.blender[N].reset() (on page 8-424)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`trigger.blender[N].reset()`

N	The trigger event blender (1 or 2)

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable`
- `trigger.blender[N].stimulus[M]`

It also clears `trigger.blender[N].overrun`.

Example

`trigger.blender[1].reset()`	Resets the trigger blender 1 settings to factory defaults.

Also see

trigger.blender[N].orenable (on page 8-423)
trigger.blender[N].overrun (on page 8-424)
trigger.blender[N].stimulus[M] (on page 8-425)

trigger.blender[N].stimulus[M]

This attribute specifies which events trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup Trigger blender N reset	Create configuration script Save setup	0

Usage

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

eventID	The event that triggers the blender action; see **Details**
N	An integer representing the trigger event blender (1 or 2)
M	An integer representing the stimulus index (1 to 4)

Details

There are four acceptors that can each select a different event. The *eventID* parameter can be the event ID of any trigger event.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID`	A channel trigger event starts the scan.
`digio.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID`	The trigger key on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID`	A delay expired.
`tsplink.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY`	Scan ready event.
`scan.trigger.EVENT_SCAN_START`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP`	Scan complete event.
`scan.trigger.EVENT_IDLE`	Idle event.
`schedule.alarm[N].EVENT_ID`	A scan starts when alarm *N* fires.

Example

```
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 8-424)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.blender[*N*].wait(*timeout*)

triggered	Trigger detection indication for blender
N	The trigger blender (1 or 2) on which to wait
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.trigger[3].mode = digio.TRIG_FALLING
digio.trigger[5].mode = digio.TRIG_FALLING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens either on line 3 or 5.

```
print(trigger.blender[1].wait(3))
```

Wait three seconds while checking if trigger blender 1 event has occurred.

If the blender trigger event has happened, then `true` is output. If the trigger event has not happened, then `false` is output after the timeout expires.

Also see

trigger.blender[N].clear() (on page 8-422)

trigger.clear()

This function clears the command interface trigger event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

The trigger event detector indicates if a trigger event has been detected since the last `trigger.wait()` call. `trigger.clear()` clears the trigger event detector and discards the history of command interface trigger events.

Also see

trigger.wait() (on page 8-435)

trigger.EVENT_ID

This constant contains the command interface trigger event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.EVENT_ID
```

eventID	The event ID for the command interface triggers

Details

You can set the stimulus of any trigger object to the value of this constant to have the trigger object respond to command interface trigger events.

Example

`scan.trigger.channel.stimulus = trigger.EVENT_ID`	Sets the trigger stimulus of the channel event detector to command an interface trigger event.

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 to clear (1 to 4)

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

`trigger.timer[1].clear()`	Clears trigger timer 1.

Also see

trigger.timer[N].count (on page 8-429)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Create configuration script Save setup	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger (0 to 1,048,575)
N	A trigger timer number (1 to 4)

Details

If *count* is set to a number greater than 1, the timer automatically starts the next delay at expiration of the previous delay.

Set *count* to zero (0) to cause the timer to generate trigger events indefinitely.

Example

`print(trigger.timer[1].count)`	Read trigger count for timer number 1.

Also see

trigger.timer[N].clear() (on page 8-429)
trigger.timer[N].reset() (on page 8-433)

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	Instrument reset Recall setup Trigger timer N reset	Create configuration script Save setup	10e-6 (10 µs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds (1.00e-09 to 100,000)
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered, it uses this delay period.

Assigning a value to this attribute is equivalent to:

`trigger.timer[N].delaylist = {interval}`

This creates a delay list of one value.

Reading this attribute returns the delay interval that will be used the next time the timer is triggered.

Example

`trigger.timer[1].delay = 50e-6`	Set the trigger timer 1 to delay for 50 µs.

Also see

trigger.timer[N].reset() (on page 8-433)

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	Instrument reset Recall setup Trigger timer N reset	Create configuration script Save setup	{10e-6}

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

intervals	Table of delay intervals in seconds
N	Trigger timer number (1 to 4)

Details

Each time the timer is triggered, it uses the next delay period from the array. The default value is an array with one value of 10 µs.

After all elements in the array have been used, the delays restart at the beginning of the list.

If the array contains more than one element, the average of the delay intervals in the list must be ≥ 50 µs.

Example

`trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6}`	Set a delay list on trigger timer 3 with three delays (50 µs, 100 µs, and 150 µs).
`DelayList = trigger.timer[3].delaylist` `for x = 1, table.getn(DelayList) do` `   print(DelayList[x])` `end`	Read the delay list on trigger timer 3. Output (assuming the delay list was set to 50 µs, 100 µs, and 150 µs): `5.000000000e-05` `1.000000000e-04` `1.500000000e-04`

Also see

trigger.timer[N].reset() (on page 8-433)

trigger.timer[N].EVENT_ID

This constant specifies the trigger timer event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

`eventID = trigger.timer[N].EVENT_ID`

eventID	The trigger event number
N	Trigger timer number (1 to 4)

Details

This constant is an identification number that identifies events generated by this timer.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to events from this timer.

Example

`scan.trigger.channel.stimulus = trigger.timer[2].EVENT_ID`	Sets the trigger stimulus of the channel event detector to trigger timer 2 event.

Also see

None

trigger.timer[N].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	Instrument reset Recall setup Trigger timer N clear Trigger timer N reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

overrun	Trigger overrun state
N	Trigger timer number (1 to 4)

Details

This attribute indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Example

`print(trigger.timer[1].overrun)`	If an event was ignored, the output is `true`. If the event was not ignored, the output is `false`.

Also see

trigger.timer[N].reset() (on page 8-433)

trigger.timer[N].passthrough

This attribute enables or disables the timer trigger pass-through mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup Trigger timer N reset	Create configuration script Save setup	false (disabled)

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

passthrough	The state of pass-through mode; set to to one of the following values: `true`: Enabled `false`: Disabled
N	Trigger timer number (1 to 4)

Details

When pass-through mode is enabled, triggers are passed through immediately and initiate the delay. When disabled, a trigger only initiates a delay.

Example

`trigger.timer[1].passthrough = true`	Enables pass-through mode on trigger timer 1.

Also see

trigger.timer[N].reset() (on page 8-433)

trigger.timer[N].reset()

This function resets some of the trigger timer settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.timer[N].reset()
```

N	Trigger timer number (1 to 4)

Details

The `trigger.timer[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.timer[N].count`
- `trigger.timer[N].delay`
- `trigger.timer[N].delaylist`
- `trigger.timer[N].passthrough`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

`trigger.timer[1].reset()`	Resets the attributes associated with timer 1 back to factory default values.

Also see

trigger.timer[N].count (on page 8-429)
trigger.timer[N].delay (on page 8-430)
trigger.timer[N].delaylist (on page 8-430)
trigger.timer[N].overrun (on page 8-431)
trigger.timer[N].passthrough (on page 8-432)
trigger.timer[N].stimulus (on page 8-433)

trigger.timer[N].stimulus

This attribute specifies which event starts the timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup Trigger timer N reset	Create configuration script Save setup	0

Usage

```
eventID = trigger.timer[N].stimulus
trigger.timer[N].stimulus = eventID
```

eventID	The event that triggers the timer delay
N	Trigger timer number (1 to 4)

Details

The *eventID* parameter may be one of the trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID`	A channel trigger event starts the scan.
`digio.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID`	The trigger key on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID`	A delay expired.
`tsplink.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY`	Scan ready event.
`scan.trigger.EVENT_SCAN_START`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP`	Scan complete event.
`scan.trigger.EVENT_IDLE`	Idle event.
`schedule.alarm[N].EVENT_ID`	A scan starts when alarm *N* fires.

Set this attribute to the *eventID* of any trigger event to wait for that event.

Use zero (0) to disable event processing.

Example

`print(trigger.timer[1].stimulus)`	Prints the event that will start a trigger 1 timer action.

Also see

trigger.timer[N].reset() (on page 8-433)

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`triggered = trigger.timer[N].wait(timeout)`

triggered	Trigger detection indication
N	Trigger timer number (1 to 4)
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`triggered = trigger.timer[3].wait(10)` `print(triggered)`	Waits up to 10 seconds for a trigger on timer 3. If `false` is returned, no trigger was detected during the 10-second timeout. If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 8-429)

trigger.wait()

This function waits for a command interface trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`triggered = trigger.wait(timeout)`

triggered	`true`: A trigger was detected during the timeout period `false`: No triggers were detected during the timeout period
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A *TRG message is received

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

`triggered = trigger.wait(10)` `print(triggered)`	Waits up to 10 seconds for a trigger. If `false` is returned, no trigger was detected during the 10-second timeout. If `true` is returned, a trigger was detected.

Also see

trigger.clear() (on page 8-428)

tsplink.group

This attribute contains the group number of a TSP-Link node. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

groupNumber	The group number of the TSP-Link node (0 to 64)

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

Example

`tsplink.group = 3`	Assign the instrument to TSP-Link group number 3.

Also see

Using groups to manage nodes on TSP-Link network (on page 7-51)

tsplink.master

This attribute reads the node number assigned to the master node. This attribute is not available on the Models 2604A/2614A/2634A.

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

Details

After doing a TSP-Link reset (`tsplink.reset()`), use this attribute to access the node number of the master in a set of instruments connected over TSP-Link.

Example

`LinkMaster = tsplink.master`	Store the TSP-Link master node number in a variable called LinkMaster.

Also see

tsplink.reset() (on page 8-439)

tsplink.node

This attribute defines the node number. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure (1 to 64)

Details

This attribute sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.reset()` from an earlier TSP-Link instrument is executed on any node in the system.

Each node connected to the TSP-Link system must be assigned a different node number.

Example

`tsplink.node = 3`	Sets the TSP-Link node for this instrument to number 3.

Also see

tsplink.reset() (on page 8-439) tsplink.reset() (on page 8-439)
tsplink.state (on page 8-440)

tsplink.readbit()

This function reads the state of a TSP-Link synchronization line. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readbit(N)
```

data	The state of the synchronization line
N	The trigger line (1 to 3)

Details

Returns a value of zero (0) if the line is low and 1 if the line is high.

Example

`data = tsplink.readbit(3)` `print(data)`	Assume line 3 is set high, and it is then read. Output: `1.000000e+00`

Also see

tsplink.readport() (on page 8-438)
tsplink.writebit() (on page 8-449)

tsplink.readport()

This function reads the TSP-Link synchronization lines as a digital I/O port. This function is not available on the Models 2604A/2614A/2634A.

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

TSP-Link synchronization lines (on page 3-48)
tsplink.readbit() (on page 8-438)
tsplink.writebit() (on page 8-449) tsplink.readbit() (on page 8-438)
tsplink.writebit() (on page 8-449)
tsplink.writeport() (on page 8-449)

tsplink.reset()

This function initializes (resets) all nodes (instruments) in the TSP-Link system. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.reset()
nodesFound = tsplink.reset(expectedNodes)
```

nodesFound	The number of nodes actually found on the system
expectedNodes	The number of nodes expected on the system (1 to 64)

Details

This function erases all information regarding other nodes connected on the TSP-Link system and regenerates the system configuration. This function must be called at least once before any remote nodes can be accessed. If the node number for any instrument is changed, the TSP-Link must be reset again.

If *`expectedNodes`* is not given, this function generates an error if no other nodes are found on the TSP-Link network.

If *`nodesFound`* is less than *`expectedNodes`*, an error is generated. Note that the node on which the command is running is counted as a node. For example, giving an expected node count of 1 will not generate any errors, even if there are no other nodes on the TSP-Link network.

Also returns the number of nodes found.

Example

`nodesFound = tsplink.reset(2)` `print("Nodes found = " .. nodesFound)`	Perform a TSP-Link reset and indicate how many nodes are found. Sample output if two nodes are found: `Nodes found = 2` Sample output if fewer nodes are found and if `localnode.showerrors = 1`: `1219, TSP-Link found fewer nodes than expected` `Nodes found = 1`

Also see

localnode.showerrors (on page 8-300)
tsplink.node (on page 8-437)
tsplink.state (on page 8-440)

tsplink.state

This attribute describes the TSP-Link online state. This attribute is not available on the Models 2604A/2614A/2634A.

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.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 8-437)
tsplink.reset() (on page 8-439) tsplink.reset() (on page 8-439)

tsplink.trigger[N].assert()

This function simulates the occurrence of the trigger and generates the corresponding event ID. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].assert()
```

N	The trigger line (1 to 3)

Details

The set pulse width determines how long the trigger is asserted.

Example

`tsplink.trigger[2].assert()`	Asserts trigger on trigger line 2.

Also see

tsplink.trigger[N].clear() (on page 8-441)
tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].pulsewidth (on page 8-445)
tsplink.trigger[N].release() (on page 8-445)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

tsplink.trigger[N].clear()

This function clears the event detector for a trigger. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].clear()
```

N	The trigger line (1 to 3)

Details

The trigger event detector enters the detected state when an event is detected.
`tsplink.trigger[N].clear()` clears a trigger event detector, discards the history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.

Example

`tsplink.trigger[2].clear()`	Clears trigger event on synchronization line 2.

Also see

tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].release() (on page 8-445)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

tsplink.trigger[N].EVENT_ID

This constant identifies the number that is used for the trigger events. This constant is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = tsplink.trigger[*N*].EVENT_ID

eventID	The trigger event number
N	The trigger line (1 to 3)

Details

This number is used by the TSP-Link trigger line when it detects an input trigger.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this line.

Example

`trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID`	Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

tsplink.trigger[N].mode

This attribute defines the trigger operation and detection mode. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup TSP-Link trigger N reset	Create configuration script Save setup	0 (tsplink.TRIG_BYPASS)

Usage

```
mode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = mode
```

mode	The trigger mode
N	The trigger line (1 to 3)

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

The setting for *mode* can be one of the following values:

Mode	Number value	Description
`tsplink.TRIG_BYPASS`	0	Allows direct control of the line as a digital I/O line.
`tsplink.TRIG_FALLING`	1	Detects falling-edge triggers as input. Asserts a TTL-low pulse for output.
`tsplink.TRIG_RISING`	2	If the programmed state of the line is high, the `tsplink.TRIG_RISING` mode behaves similarly to `tsplink.TRIG_RISINGA`. If the programmed state of the line is low, the `tsplink.TRIG_RISING` mode behaves similarly to `tsplink.TRIG_RISINGM`. Use `tsplink.TRIG_RISINGA` if the line is in the high output state. Use `tsplink.TRIG_RISINGM` if the line is in the low output state.
`tsplink.TRIG_EITHER`	3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
`tsplink.TRIG_SYNCHRONOUSA`	4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low.
`tsplink.TRIG_SYNCHRONOUS`	5	Detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserts a TTL-low pulse as an output trigger.
`tsplink.TRIG_SYNCHRONOUSM`	6	Detects rising-edge triggers as an input. Asserts a TTL-low pulse for output.
`tsplink.TRIG_RISINGA`	7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
`tsplink.TRIG_RISINGM`	8	Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

When programmed to any mode except `tsplink.TRIG_BYPASS`, the output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

When the trigger mode is set to `tsplink.TRIG_RISING`, the user-specified output state of the line is examined. If the output state selected when the mode is changed is high, the actual mode used will be `tsplink.TRIG_RISINGA`. If the output state selected when the mode is changed is low, the actual mode used will be `tsplink.TRIG_RISINGM`.

`mode` stores the trigger mode as a numeric value when the attribute is read.

To control the line state, use the `tsplink.TRIG_BYPASS` mode with the `tsplink.writebit()` and the `tsplink.writeport()` commands.

Example

`tsplink.trigger[3].mode = tsplink.TRIG_RISINGM`	Sets the trigger mode for synchronization line 3 to `tsplink.TRIG_RISINGM`.

Also see

digio.writebit() (on page 8-129)
digio.writeport() (on page 8-130)
tsplink.trigger[N].assert() (on page 8-441)
tsplink.trigger[N].clear() (on page 8-441)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].release() (on page 8-445)
tsplink.trigger[N].reset() (on page 8-446)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

tsplink.trigger[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Instrument reset Recall setup TSP-Link trigger N clear TSP-Link trigger N reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

`overrun`	Trigger overrun state
`N`	The trigger line (1 to 3)

Details

Indicates that an event was ignored because the event detector was in the detected state when the event was detected.

Indicates the overrun state of the event detector built into the line itself.

It does not indicate whether an overrun occurred in any other part of the trigger model or in any other detector that is monitoring the event.

It does not indicate output trigger overrun.

Example

`print(tsplink.trigger[1].overrun)`	If an event was ignored, displays `true`; if an event was not ignored, displays `false`.

Also see

tsplink.trigger[N].assert() (on page 8-441)
tsplink.trigger[N].clear() (on page 8-441)
tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].release() (on page 8-445)
tsplink.trigger[N].reset() (on page 8-446)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

tsplink.trigger[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Reset Recall setup TSP-Link trigger N reset	Create configuration script Save setup	10e-6 (10 µs)

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

width	The pulse width (in seconds)
N	The trigger line (1 to 3)

Details

Setting the pulse width to 0 (seconds) asserts the trigger indefinitely.

Example

`tsplink.trigger[3].pulsewidth = 20e-6`	Sets pulse width for trigger line 3 to 20 µs.

Also see

tsplink.trigger[N].release() (on page 8-445)

tsplink.trigger[N].release()

This function releases a latched trigger on the given TSP-Link trigger line. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].release()
```

N	The trigger line (1 to 3)

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

`tsplink.trigger[3].release()`	Releases trigger line 3.

Also see

tsplink.trigger[N].assert() (on page 8-441)
tsplink.trigger[N].clear() (on page 8-441)
tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].pulsewidth (on page 8-445)
tsplink.trigger[N].stimulus (on page 8-447)
tsplink.trigger[N].wait() (on page 8-448)

tsplink.trigger[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].reset()
```

N	The trigger line (1 to 3)

Details

The `tsplink.trigger[N].reset()` function resets the following attributes to their factory defaults:

- `tsplink.trigger[N].mode`
- `tsplink.trigger[N].stimulus`
- `tsplink.trigger[N].pulsewidth`

This also clears `tsplink.trigger[N].overrun`.

Example

`tsplink.trigger[3].reset()`	Resets TSP-Link trigger line 3 attributes back to factory default values.

Also see

tsplink.trigger[N].mode (on page 8-443)
tsplink.trigger[N].overrun (on page 8-444)
tsplink.trigger[N].pulsewidth (on page 8-445)
tsplink.trigger[N].stimulus (on page 8-447)

tsplink.trigger[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	Reset Recall setup TSP-Link trigger N reset	Create configuration script Save setup	0

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

eventID	The event identifier for the triggering event
N	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to zero (0).

Do not use this attribute when triggering under script control. Use `tsplink.trigger[N].assert()` instead.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

Trigger event IDs

Trigger event ID	Description
`channel.trigger[N].EVENT_ID`	A channel trigger event starts the scan.
`digio.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line.
`display.trigger.EVENT_ID`	The trigger key on the front panel is pressed.
`dmm.trigger.EVENT_LIMIT1_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 1.
`dmm.trigger.EVENT_LIMIT1_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 1.
`dmm.trigger.EVENT_LIMIT2_HIGH`	A DMM trigger event that indicates a measurement has exceed the high limit value on limit 2.
`dmm.trigger.EVENT_LIMIT2_LOW`	A DMM trigger event that indicates a measurement has exceed the low limit value on limit 2.
`trigger.EVENT_ID`	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event.
`trigger.blender[N].EVENT_ID`	A combination of events has occurred.
`trigger.timer[N].EVENT_ID`	A delay expired.
`tsplink.trigger[N].EVENT_ID`	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line.
`lan.trigger[N].EVENT_ID`	A LAN trigger event has occurred.
`scan.trigger.EVENT_SCAN_READY`	Scan ready event.
`scan.trigger.EVENT_SCAN_START`	Scan start event.
`scan.trigger.EVENT_CHANNEL_READY`	Channel ready event.
`scan.trigger.EVENT_MEASURE_COMP`	Measure complete event.
`scan.trigger.EVENT_SEQUENCE_COMP`	Sequence complete event.
`scan.trigger.EVENT_SCAN_COMP`	Scan complete event.
`scan.trigger.EVENT_IDLE`	Idle event.
`schedule.alarm[N].EVENT_ID`	A scan starts when alarm *N* fires.

Example

`tsplink.trigger[3].stimulus = scan.trigger.EVENT_CHANNEL_READY`
Sets the trigger stimulus of the TSP-Link trigger line 3 event detector to scan the trigger channel ready event.

Also see

tsplink.trigger[N].assert() (on page 8-441)
tsplink.trigger[N].reset() (on page 8-446)

tsplink.trigger[N].wait()

This function waits for a trigger. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = tsplink.trigger[N].wait(timeout)
```

triggered	Trigger detection indication; set to one of the following values: `true`: A trigger is detected during the timeout period `false`: A trigger is not detected during the timeout period
N	The trigger line (1 to 3)
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events were detected since the last time `tsplink.trigger[N].wait()` or `tsplink.trigger[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 = tsplink.trigger[3].wait(10)` `print(triggered)`	Waits up to 10 seconds for a trigger on TSP-Link® line 3. If `false` is returned, no trigger was detected during the 10-second timeout. If `true` is returned, a trigger was detected.

Also see

tsplink.trigger[N].clear() (on page 8-441)

tsplink.writebit()

This function sets a TSP-Link synchronization line high or low. This function is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`tsplink.writebit(N, data)`

N	The trigger line (1 to 3)
data	The value to write to the bit: • Low: 0 • High: 1

Details

Use `tsplink.writebit()` and `tsplink.writeport()` to control the output state of the trigger line when trigger operation is set to `tsplink.TRIG_BYPASS`.

If the output line is write-protected by the `tsplink.writeprotect` attribute, this command is ignored.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

`tsplink.writebit(3, 0)`	Sets trigger line 3 low (0).

Also see

tsplink.readbit() (on page 8-438)
tsplink.readport() (on page 8-438)
tsplink.writeport() (on page 8-449)
tsplink.writeprotect (on page 8-450)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines. This function is not available on the Models 2604A/2614A/2634A.

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).

Write-protected lines are not changed.Write-protected lines are not changed.

The `reset()` function does not affect the present states of the trigger lines.

Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `tsplink.TRIG_BYPASS`.Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `tsplink.TRIG_BYPASS`.

Example

`tsplink.writeport(3)`	Sets the synchronization lines 1 and 2 high (binary 011).

Also see

tsplink.readbit() (on page 8-438) tsplink.readbit() (on page 8-438)
tsplink.readport() (on page 8-438)
tsplink.writebit() (on page 8-449)
tsplink.writeprotect (on page 8-450) tsplink.writebit() (on page 8-449)
tsplink.writeprotect (on page 8-450)

tsplink.writeprotect

This attribute contains the write-protect mask that protects bits from changes by the `tsplink.writebit()` and `tsplink.writeport()` functions. This attribute is not available on the Models 2604A/2614A/2634A.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Instrument reset Recall setup	Create configuration script Saved setup	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

mask	An integer that specifies the value of the bit pattern for write-protect; set bits to 1 to write-protect the corresponding TSP-Link trigger line

Details

The binary equivalent of *mask* indicates the mask to be set for the TSP-Link trigger line. For example, a *mask* value of 5 has a binary equivalent of 101. This *mask* write-protects TSP-Link trigger lines 1 and 3.

Example

`tsplink.writeprotect = 5`	Write-protects TSP-Link trigger lines 1 and 3.

Also see

tsplink.readbit() (on page 8-438)
tsplink.readport() (on page 8-438)
tsplink.writebit() (on page 8-449)
tsplink.writeport() (on page 8-449)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

`tspnet.write(testdevice, "print([[hello]])")` `print(tspnet.readavailable(testdevice))`	Write data to a device, then print how much is available. Output: `6.00000e+00`
`tspnet.clear(testdevice)` `print(tspnet.readavailable(testdevice))`	Clear data and print how much data is available again. Output: `0.00000e+00`

Also see

tspnet.connect() (on page 8-451)
tspnet.readavailable() (on page 8-456)
tspnet.write() (on page 8-461)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect(ipAddress)
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

connectionID	The connection ID to be used as a handle in all other `tspnet` function calls
ipAddress	IP address to which to connect in a string
portNumber	Port number (default 5025)
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. If the *portNumber* is 23, the interface uses the Telnet protocol and sets appropriate termination characters to communicate with the device.

If a *portNumber* and *initString* are provided, it is assumed that the remote device is not TSP-enabled. The Series 3700A does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a *portNumber* nor an *initString* is provided, the remote device is assumed to be a Keithley Instruments TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the Series 3700A sends an `abort` command to the remote device on connection.

The Series 3700A also enables TSP prompts on the remote device and error management. The Series 3700A places remote errors from the TSP-enabled device in its own error queue and prefaces these errors with `Remote Error`, followed by an error description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show errors by changing `localnode.showerrors` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
instrumentID = tspnet.connect("192.0.2.1")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device.

Example 2

```
instrumentID = tspnet.connect("192.0.2.1", 1394,
    "*rst\r\n")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a device that is not TSP-enabled.

Also see

localnode.prompts (on page 8-296)
localnode.showerrors (on page 8-300)
tspnet.tsp.abortonconnect (on page 8-459)
tspnet.disconnect() (on page 8-452)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.disconnect(`*`connectionID`*`)`

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function disconnects the two devices by closing the connection. The *`connectionID`* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

`testID = tspnet.connect("192.0.2.0")` `-- Use the connection` `tspnet.disconnect(testID)`	Create a TSP-Net session. Close the session.

Also see

tspnet.connect() (on page 8-451)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute(connectionID, commandString)
value1 = tspnet.execute(connectionID, commandString, formatString)
value1, value2 = tspnet.execute(connectionID, commandString, formatString)
value1, ..., valuen = tspnet.execute(connectionID, commandString, formatString)
```

`connectionID`	The connection ID returned from `tspnet.connect()`
`commandString`	The command to send to the remote device
`value1`	The first value decoded from the response message
`value2`	The second value decoded from the response message
`valuen`	The nth value decoded from the response message; there is one return value for each format specifier in the format string
`...`	One or more values separated with commas
`formatString`	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The Series 3700A 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 Series 3700A suspends operation until a timeout error is generated or until the instrument responds, even if no format string is specified. The TSP prompt from the remote instrument is read and thrown away. The Series 3700A places any remotely generated errors into its error queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

`tspnet.execute(instrumentID, "runScript()")`	Command the remote device to run a script named `runScript`.

Example 2

`tspnet.termination(instrumentID, tspnet.TERM_CRLF)` `tspnet.execute(instrumentID, "*idn?")` `print("tspnet.execute returns:", tspnet.read(instrumentID))`	Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 8-451)
tspnet.read() (on page 8-455)
tspnet.termination() (on page 8-457)
tspnet.write() (on page 8-461)

tspnet.idn()

This function retrieves the response of the remote device to `*IDN?`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
idnString = tspnet.idn(connectionID)
```

idnString	The returned `*IDN?` string
connectionID	The connection ID returned from `tspnet.connect()`

Details

This function retrieves the response of the remote device to `*IDN?`.

Example

`deviceID = tspnet.connect("192.0.2.1")` `print(tspnet.idn(deviceID))` `tspnet.disconnect(deviceID)`	Assume the instrument is at IP address 192.0.2.1. The output that is produced when you connect to the instrument and read the IDN string may appear as: `KEITHLEY INSTRUMENTS INC.,MODEL 3706A,00000170,01.10h`

Also see

tspnet.connect() (on page 8-451)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The nth value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from `tspnet.connect()`
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

`%[width]s`	Read data until the specified length
`%[max width]t`	Read data until the specified length or until punctuation is found, whichever comes first
`%[max width]n`	Read data until a newline or carriage return
`%d`	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If *formatString* is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the Series 3700A 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 Series 3700A reads from a TSP-enabled remote instrument, the Series 3700A removes Test Script Processor (TSP®) prompts and places any errors it receives from the remote instrument into its own error queue. The Series 3700A prefaces errors from the remote device with "`Remote Error`," followed by the error number and error description.

Example

`tspnet.write(deviceID, "*idn?\r\n")`	Send the `"*idn?\r\n"` message to the instrument connected as `deviceID`.
`print("write/read returns:", tspnet.read(deviceID))`	Display the response that is read from `deviceID` (based on the `*idn?` message).

Also see

tspnet.connect() (on page 8-451)
tspnet.readavailable() (on page 8-456)
tspnet.timeout (on page 8-458)
tspnet.write() (on page 8-461)

tspnet.readavailable()

This function checks to see if data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`bytesAvailable = tspnet.readavailable(connectionID)`

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from `tspnet.connect()`

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")

repeat bytes = tspnet.readavailable(ID) until bytes > 0

print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that will create data.

Wait for data to be available.

Also see

tspnet.connect() (on page 8-451)
tspnet.read() (on page 8-455)

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	An enumerated value indicating the termination type: • 1 or `tspnet.TERM_LF` • 4 or `tspnet.TERM_CR` • 2 or `tspnet.TERM_CRLF` • 3 or `tspnet.TERM_LFCR`
connectionID	The connection ID returned from `tspnet.connect()`
termSequence	The termination sequence

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the *termSequence* parameter sets the termination sequence. The present termination sequence is always returned.

For the *termSequence* parameter, use the same values listed in the table above for type. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is `tspnet.TERM_LF`. For devices that are not TSP-enabled, the default is `tspnet.TERM_CRLF`.

Example

`deviceID = tspnet.connect("192.0.2.1")` `if deviceID then` `tspnet.termination(deviceID, tspnet.TERM_LF)` `end`	Sets termination type for IP address 192.0.2.1 to TERM_LF.

Also see

tspnet.connect() (on page 8-451)
tspnet.disconnect() (on page 8-452)

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	Instrument reset Recall setup	Create configuration script	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds (0.001 to 30.000)

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands will wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution, but is only accurate to the nearest 10 ms.

Example

`tspnet.timeout = 2.0`	Sets the timeout duration to two seconds.

Also see

tspnet.connect() (on page 8-451)
tspnet.execute() (on page 8-453)
tspnet.read() (on page 8-455)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other `tspnet` commands

Details

This function is appropriate only for TSP-enabled instruments.

Sends an abort command to the remote instrument.

Example

`tspnet.tsp.abort(testConnection)`	Stops remote instrument execution on `testConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Instrument reset Recall setup	Create configuration script Save setup	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	1 (enable) or 0 (disable)

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument will not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

`tspnet.tsp.abortonconnect = 0`	Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

tspnet.connect() (on page 8-451)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, name)
table = tspnet.tsp.rbtablecopy(connectionID, name, startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other `tspnet` commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

`times = tspnet.tsp.rbtablecopy(testTspdevice, "testRemotebuffername.timestamps", 1, 3)` `print(times)`	Copy the specified timestamps table for items 1 through 3, then display the table. Sample output: `01/01/2011 10:10:10.0000013,01/01/2011 10:10:10.0000233,01/01/2011 10:10:10.0000576`

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.tsp.runscript(connectionID, script)
tspnet.tsp.runscript(connectionID, name, script)
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions at a later time.

If no name is specified, the script is loaded as the anonymous script.

Example

```
tspnet.tsp.runscript(myconnection, "mytest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]])")
```

Load and run a script entitled `mytest` on the TSP-enabled instrument connected with `myconnection`.

Also see

tspnet.execute() (on page 8-453)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.write(connectionID, inputString)
```

connectionID	The connection ID returned from `tspnet.connect()`
inputString	The string to be written

Details

The `tspnet.write()` function sends *`inputString`* to the remote instrument. It does not wait for command completion on the remote instrument.

The Series 3700A sends *`inputString`* to the remote instrument exactly as indicated. The *`inputString`* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

`tspnet.write(myID, "runscript()\r\n")`	Commands the remote instrument to execute a command or script named "runscript()" on a remote device identified in the system as `myID`.

Also see

tspnet.connect() (on page 8-451)
tspnet.read() (on page 8-455)

upgrade.previous()

This function returns to a previous version of the Model 3706A firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.previous()
```

Details

This function allows you to revert to an earlier version of the firmware.

When you send this function, the instrument searches the flash drive that is inserted in the front-panel USB port for an upgrade file. If the file is found, the instrument performs the upgrade. An error is returned if an upgrade file is not found.

Also see

Upgrading the firmware (on page A-6)
upgrade.unit() (on page 8-463)

upgrade.unit()

This function upgrades the Model 3706A firmware.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
upgrade.unit()
```

Details

When `upgrade.unit()` is used, the firmware is only loaded if the version of the firmware component is newer than the existing version. If the version is older or at the same revision level, it is not upgraded.

When you send this function, the instrument searches the flash drive that is inserted in the front-panel USB port for an upgrade file. If the file is found, the instrument verifies that the file is a newer version. If the version is older or at the same revision level, it is not upgraded. If it is a newer version, the instrument performs the upgrade. An error is returned if no upgrade file is found.

Also see

Upgrading the firmware (on page A-6)
upgrade.previous() (on page 8-462)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
userstring.add(name, value)
```

name	The name of the string; the key of the key-value pair
value	The string to associate with *name*; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

`userstring.add("assetnumber", "236")` `userstring.add("product", "Widgets")` `userstring.add("contact", "John Doe")` `for name in userstring.catalog() do` `   print(name .. " = " ..` `      userstring.get(name))` `end`	Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.

Also see

userstring.catalog() (on page 8-464)
userstring.delete() (on page 8-465)
userstring.get() (on page 8-465)

userstring.catalog()

This function creates an iterator for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do body end
```

name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The catalog provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

`for name in userstring.catalog() do` `   userstring.delete(name)` `end`	Deletes all user-defined strings in nonvolatile memory.

Example 2

`for name in userstring.catalog() do` `   print(name .. " = " ..` `      userstring.get(name))` `end`	Prints all `userstring` key-value pairs. Output: `product = Widgets` `assetnumber = 236` `contact = John Doe` The above output lists the user-defined strings added in the example for the `userstring.add()` function. Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 8-463)
userstring.delete() (on page 8-465)
userstring.get() (on page 8-465)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
userstring.delete(name)
```

name	The name (key) of the key-value pair of the user-defined string to delete

Details

This function deletes the string that is associated with *name* from nonvolatile memory.

Example

`userstring.delete("assetnumber")` `userstring.delete("product")` `userstring.delete("contact")`	Deletes the user-defined strings associated with the `assetnumber`, `product`, and `contact` names.

Also see

userstring.add() (on page 8-463)
userstring.catalog() (on page 8-464)
userstring.get() (on page 8-465)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = userstring.get(name)
```

value	The value of the user-defined string key-value pair
name	The name (key) of the user-defined string

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

`value = userstring.get("assetnumber")` `print(value)`	Read the value associated with a user-defined string named `"assetnumber"`. Store it in a variable called value, then print the variable value. Output: `236`

Also see

userstring.add() (on page 8-463)
userstring.catalog() (on page 8-464)
userstring.delete() (on page 8-465)

waitcomplete()

This function waits for all overlapped commands in a specified group to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies which TSP-Link group on which to wait

Details

This function will wait for all previously started overlapped commands to complete.

Currently, the Series 3700A has no overlapped commands implemented. However, other TSP-enabled products, such as the Series 2600A System SourceMeter® Instruments, have overlapped commands. Therefore, when the Series 3700A is a TSP master to a subordinate device with overlapped commands, use this function to wait until all overlapped operations are completed.

A group number may only be specified when this node is the master node.

If no *group* is specified, the local group is used.

If zero (0) is specified for the *group*, this function waits for all nodes in the system.

NOTE

Any nodes that are not assigned to a group (group number is 0) are part of the master node's group.

Example 1

`waitcomplete()`	Waits for all nodes in the local group.

Example 2

`waitcomplete(G)`	Waits for all nodes in group G.

Example 3

`waitcomplete(0)`	Waits for all nodes on the TSP-Link network.

Also see

None

Section 9

Troubleshooting guide

In this section:

Contacting support

If you have any questions after reviewing this information, please contact your local Keithley Instruments representative or call Keithley Instruments corporate headquarters (toll-free inside the U.S. and Canada only) at 1-888-KEITHLEY (1-888-534-8453), or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (*http://www.keithley.com*).

When contacting Keithley, please have ready:

- The serial number of the instrument.
- The firmware revision of the instrument.
- The model and firmware revision of all installed cards.

When you call, have the information available, and, if possible, be near the instrument.

Locating serial number or firmware revision

The serial number is on the rear panel of the instrument. You can also use the front panel **MENU** option to display the serial number and firmware version.

To display serial number or firmware revision on the front panel:

1. If the Series 3700A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Use the navigation wheel ◷ to scroll to the **UNIT-INFO** menu.
4. Press the **ENTER** key.

On the UNIT INFORMATION menu, scroll to the **SERIAL#** or **FIRMWARE** option and press the **ENTER** key. The Series 3700A serial number is displayed.

Locating information on the installed cards

To identify installed switching cards from the front panel:

Press the **SLOT** key to scroll through the model numbers, descriptions, and firmware revisions of the installed switching cards.

To identify installed switching cards from the web interface:

1. Select the **Unit** page.
2. In the Report area, select the slots that you want information about.
3. Select **Firmware Revision**.
4. Click **Generate Report**. Information about the cards in the slots is displayed below the button.

To identify installed switching cards from the remote command interface:

Use `print(slot[X].idn)`to query and identify installed switching cards:

```
print(slot[X].idn)
```

Where: *X* = slot number (from 1 to 6)

Example

To get a list of all switching cards installed in the slots of a Series 3700A, send the following command over the remote command interface:

```
for x=1,6 do print (slot[x].idn) end
```

The response will be similar to the following:

```
3722, Dual 1x48 Multiplexer, 01.00a, <Module Serial Number>
3721, Dual 1x20 Multiplexer, 01.02a, <Module Serial Number>
Empty Slot
Empty Slot
Empty Slot
Empty Slot
```

USB troubleshooting

This section provides information checks you can perform if the USB communication with the instrument is not working.

Check driver for the USB Test and Measurement Device

1. Open the Device Manager.

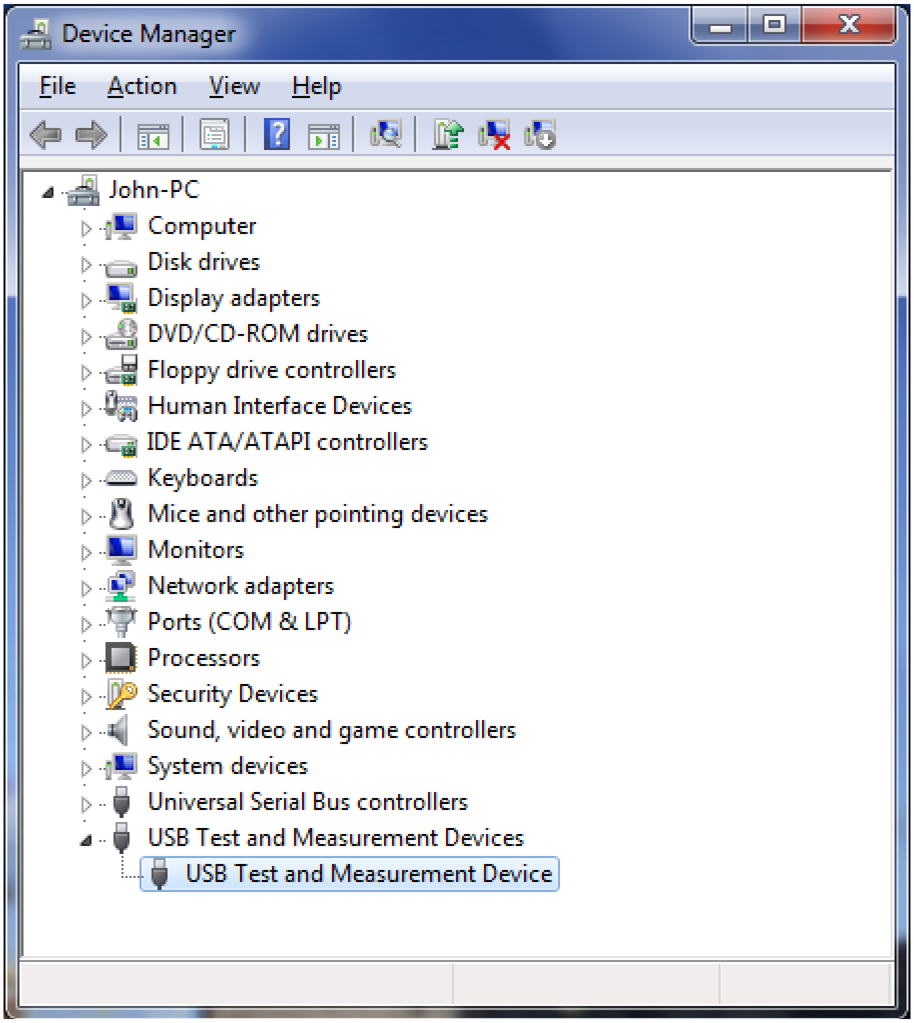
Quick Tip

From the Start menu, you can enter Devmgmt.msc in the Run box or the Windows 7 search box to start Device Manager.

2. Under USB Test and Measurement Devices, look for USB Test and Measurement Device.

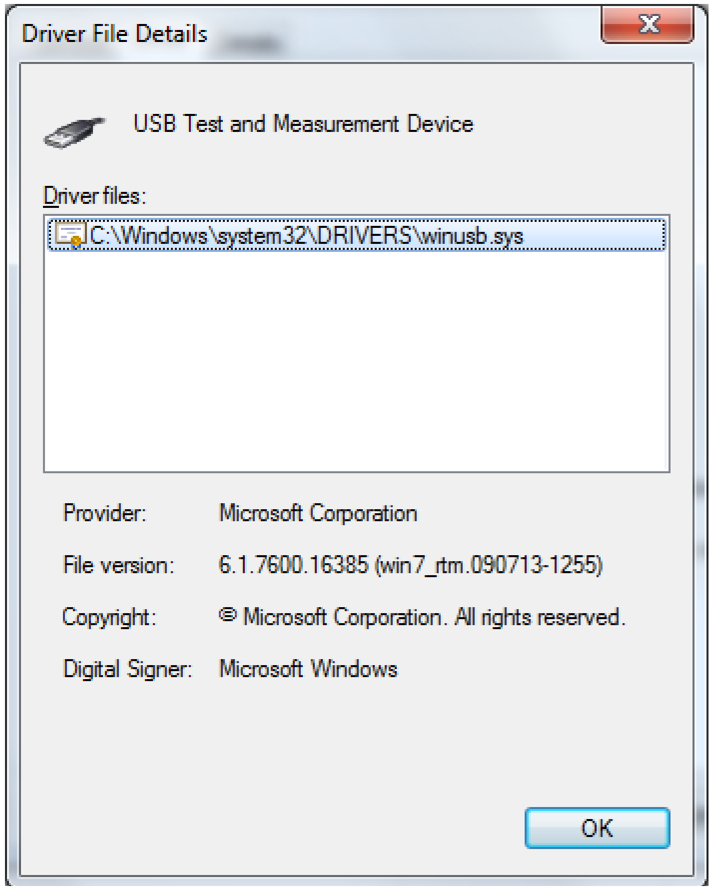
   If the device is not there, either VISA is not installed or the instrument is not plugged in and switched on.

Figure 123: Device Manager dialog box showing USB Test and Measurement Device



3. Right-click the device.
4. Select Properties.
5. Select the Driver tab.
6. Click **Driver Details**.
7. Verify that the device driver is the winusb.sys. driver from Microsoft.

Figure 124: Driver File Details dialog box



8. If the incorrect driver is installed, click **OK**.
9. On the Driver tab, click **Update Driver**.
10. Browse for the driver; select the C:\windows\inf folder and you should see the winusb.inf file. Select this and make sure the driver is now in use.
11. If this does not work, uninstall VISA, unplug the instrument and follow the steps to reinstall VISA in the section Modifying, repairing, or removing Keithley I/O Layer software (on page 2-72).

Troubleshooting GPIB interfaces

If the hardware is not recognized by the computer:

1. Uninstall the software drivers.
2. Reboot the computer.
3. Check for newer drivers on the vendor's website. Check that the drivers are valid for the operating system you have and any updates that might be necessary. This information is typically found in the readme file that comes with the drivers.
4. Install software drivers.
5. Reboot the computer.
6. Plug in the hardware.

If it is still not recognized, you can try a different computer using a different operating system to rule out operating system issues.

If this does not resolve the issue, contact the vendor of the GPIB controller for assistance.

Timeout errors

If your GPIB controller is recognized by the operating system, but you get a timeout error when you try to communicate with the instrument, check the following:

1. Confirm that the GPIB address you assigned to the instrument is unique and between the range of 0 to 30. It should not be 0 or 21 because they are common controller addresses.
2. Check cabling connection. GPIB cables are heavy and can fall out of the connectors if they are not screwed in securely.
3. Substitute cables to verify cable integrity. For example, if you can send and receive ASCII text, but you cannot do a binary transfer, check your program and the decoding of the binary data. If that does not resolve the problem, try another cable. ASCII text only uses seven data lines in the cable; the binary transfer requires all eight lines.

Troubleshooting LAN interfaces

This section provides information on troubleshooting LAN interfaces.

For detailed information on setting up remote interfaces see Communications interfaces (see "Remote communication interfaces" on page 2-53).

Verify connections and settings

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

- Verify that the crossover cable is in the correct LAN port on the instrument. Do not connect to one of the TSP-Link ports.
- Verify that the crossover cable is in the correct port on the computer. The Ethernet port of a laptop may be disabled while the computer is in a docking station.
- Verify that the correct Ethernet card configuration information was used during the setup procedure.
- Verify that the computer's network card is enabled.
- Verify the instrument IP address is compatible with the IP address on the computer.
- Verify the instrument subnet mask address is the same as the computer's subnet mask address.
- Turn the instrument power off, and then on.
- Reboot the computer.

Use Ping to test the connection

Ping is a computer network administration utility that you can use to test whether a particular host can be reached across an Internet Protocol (IP) network. It also measures the round-trip time for packets sent from the local host to a destination computer, including the local host's own interfaces.

To run Ping:

1. From the Windows Start menu, type cmd in the Run box or Search box. The Command window is displayed.
2. At the > prompt, type ping followed by the IP address. For example:

```
ping 169.254.52.51
```

Beware that some network devices, especially LXI instruments, can disable the ping response to prevent denial of service attacks. This prevents hackers from pinging your instrument indefinitely, which causes the instrument to become so busy it cannot respond to a web browser or instrument driver.

If you cannot ping an instrument from the computer, you will not be able to communicate with the instrument. You will need to check the LAN settings from the front panel of the instrument to see if they match the configuration of your network.

If you can ping your instrument, you should be able to bring up the web page in the instrument from a browser by typing the IP address in the address (URL) field.

Open ports on firewalls

A firewall is a part of a computer system or network that is designed to block unauthorized access while permitting authorized communications. It is a device or set of devices that are configured to permit or deny applications based on a set of rules and other criteria.

If you have a firewall in the network between your computer and the instrument, you need to make sure the following ports are opened for UDP and TCP packets:

- Port 80: Web server. This is normally open.
- Port 1024: VXI-11 connection for sending and receiving commands from the instrument.
- Port 5025: Raw socket connection for sending and receiving commands from the instrument.

Web page problems

All LXI instruments have a web server. The LAN configuration information on these pages is mandated by the LXI consortium. For Keithley's LXI instruments, the standard LXI pages use standard HTML.

The added value pages that Keithley has added to control the instruments use Java. If Java is not installed when you select one of these instrument-specific web pages, the web page prompts you to install it. To do this, your computer must have access to the Internet so it can access the web browser plug-in Sun Java Runtime Environment Version 6 or higher. Installation files are available at the Java download site (*http://www.java.com/en/download/manual.jsp*).

When you connect to the instrument web page for the first time, several things can happen:

- If the security settings are high, scripting might be disabled and the browser will prompt you to enable ActiveX and scripting.
- If Java is not installed, the browser will prompt you to install it and provide a link to the download. If you do not have an Internet connection, you must download it elsewhere and install it on the computer that it connected to the instrument.
- When the Java applet from the instrument gets downloaded into the browser it will ask you if you trust this active content from Keithley Instruments. Select Yes.

If you have resolved the problems, the instrument control pages should work and if you try to perform an action, such as closing a relay, you are prompted for the password (the default is "admin").

NOTE

If you update the firmware for the instrument using the web page (not available for all instruments), you need to flush the browser cache so that a fresh Java Applet gets downloaded the next time you access the web page.

LXI LAN status indicator

Most LAN network interface cards have two LEDs, one that indicates LAN traffic and one that designates the LAN speed (10 MBits, 100 MBits, 1 GBits) through the color of the LED. LXI goes one level higher than this and states that all LXI compliant devices need a LAN status indicator. This can be an LED or an indicator on a display. It shows if the instrument has a valid IP address or is in a fault state.

When diagnosing a LAN connection issue with an LXI instrument, see if the LAN status indicator is signaling a valid or fault condition. If there is an error, you cannot communicate with the instrument through the LAN connection. In this case, you need to check the LAN parameter settings from the front panel of the instrument. Make sure if you change a LAN setting through the front panel that you select the “Apply LAN Settings” for the changes to take affect.

Initialize the LAN configuration

The LXI specification mandates that all instruments that conform to LXI need a LAN reset mechanism.This can be a recessed switch or a menu option on the front panel that will put all the LAN settings back to known defaults. If you cannot communicate with your instrument, perform this reset.

If you perform a reset, the instrument is returned to DHCP and Auto-IP enabled. If you set your computer to match, you should be able to use a discovery tool to determine the IP address and communicate with the instrument again. Also check the LAN status indicator to verify that there are no faults.

Install LXI Discovery Browser software on your computer

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

The Keithley LXI Discovery Browser is available on the instrument CD. It is also available on the Keithley Instruments website (*http://www.keithley.com*).

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

1. Select the **Support** tab.
2. In the model number box, type **3706A**.
3. From the list, select **Software** and click the search icon. A list of software applications for the instrument is displayed.
4. See the readme file included with the application for more information.

For more information about the LXI Consortium, see the LXI Consortium website (*http://www.lxistandard.org/*).

Communicate using VISA communicator

There are several interactive communication utilities that you can use to communicate with LAN instruments:

- The KIOL installs the Keithley Communicator.
- NI VISA (full version) installs the NI VISA Interactive Control utility, which can also be launched from NI-MAX.
- Agilent has a similar utility called Interactive IO that gets installed with their IO Libraries Suite.

All these utilities require you to enter the VISA resource string for your instrument. See Communicate with the instrument (on page 2-54) for more information on the VISA resource string formats.

HyperTerminal, which comes with Microsoft Windows, also allows you to connect to the raw socket port of the instrument.

WireShark

WireShark is an open source LAN packet sniffer. You can run it to spy on all the packets going across a network. It allows you to filter what you spy on so that you can narrow the content down to just what you are interested in. For example, you could check just web page packets (http) or all packets being sent by a device on a certain IP address.

See the WireShark documentation for information. WireShark can be downloaded from www.wireshark.org (*http://www.wireshark.org*).

Testing the display, keys, and channel matrix

You can test operation of the keys, display, and channel matrix from the front panel of the instrument.

Verify front panel key operation

You can verify that the instrument is properly reading front panel key presses.

To verify key operation:

1. From the front panel, select **MAIN MENU > DISPLAY > TEST > KEYS**. The message "No keys pressed" is displayed.
2. Press a key. The name of the key is displayed. For a list of key values, see display.sendkey() (on page 8-144).
3. Press **EXIT (LOCAL)** twice to return to the menu.

Verify display operation

You can verify that all the pixels on the vacuum fluorescent display (VFD) are working.

To verify VFD operation:

1. From the front panel, select **MAIN MENU > DISPLAY > TEST > DISPLAY-PATTERNS**. A pattern is displayed.
2. Press the navigation wheel to display the next pattern.
3. When you have viewed the patterns, press **EXIT** to return to the menu.

Update drivers

For the latest drivers and additional support information, see the Keithley Instruments support website.

To see what drivers are available for your instrument:

1. Go to the Keithley Instruments support website (*http://www.keithley.com/support*).
2. Enter the model number of your instrument.
3. Select Software Driver from the list.

For LabVIEW™, you can also go to National Instrument's website and search their instrument driver database.

Error and status messages

Introduction

This section includes information on error levels, how to read errors, and a complete listing of error messages.

Error summary

Error and status messages are assigned a level of severity, as listed in the table below.

Severity level descriptions		
Number	**Level**	**Description**
0	**Informational**	Indicates that there are no entries in the queue
10	**Informational**	Indicates a status message or minor error
20	**Recoverable**	Indicates possible invalid user input; operation continues but action should be taken to correct the error
30	**Serious**	Indicates a serious error that may require technical assistance, such as corrupted data
40	**Fatal**	Instrument is not operational

Effects of errors on scripts

Most errors will not abort a running script. The only time a script is aborted is when a Lua runtime error (error number –286) is detected.

Runtime errors are caused by actions such as trying to index into a variable that is not a table.

Syntax errors (error number -285) in a script or command will not abort the script, but will prevent the script or command from being executed.

Retrieving errors

When errors occur, the error messages are placed in the error queue. Use error queue commands to request error message information. For example, the following commands request the next complete error information from the error queue and return the code, message, severity, and node for that error:

```
errorCode, message, severity, errorNode = errorqueue.next()
print(errorCode, message, severity, errorNode)
```

The following table lists the commands associated with the error queue.

Remote commands associated with the error queue	
Command	**Description**
errorqueue.clear() (on page 8-246)	Clear error queue of all errors
errorqueue.count (on page 8-246)	Number of messages in the error queue
errorqueue.next() (on page 8-246)	Request next error message from queue

Error and status message list

Error and status messages

Error number	Error level	Error message
-430	RECOVERABLE	Query deadlocked
-420	RECOVERABLE	Query unterminated
-410	RECOVERABLE	Query interrupted
-363	RECOVERABLE	Input buffer over-run
-360	RECOVERABLE	Communication error
-350	RECOVERABLE	Queue overflow
-315	RECOVERABLE	Configuration memory lost
-314	RECOVERABLE	Save/recall memory lost
-292	RECOVERABLE	Referenced name does not exist
-286	RECOVERABLE	TSP runtime error
-285	RECOVERABLE	Program syntax
-282	RECOVERABLE	Illegal program name
-281	RECOVERABLE	Cannot create program
-225	RECOVERABLE	Out of memory or TSP memory allocation error
-224	RECOVERABLE	Illegal parameter value
-223	RECOVERABLE	Too much data
-222	RECOVERABLE	Parameter data out of range
-221	RECOVERABLE	Settings conflict
-220	RECOVERABLE	Parameter
-203	RECOVERABLE	Command protected
-200	RECOVERABLE	Execution error
-154	RECOVERABLE	String too long
-151	RECOVERABLE	Invalid string data
-144	RECOVERABLE	Character data too long
-141	RECOVERABLE	Invalid character data
-140	RECOVERABLE	Character data error
-121	RECOVERABLE	Invalid character in number
-120	RECOVERABLE	Numeric data
-109	RECOVERABLE	Missing parameter
-108	RECOVERABLE	Parameter not allowed
-105	RECOVERABLE	Trigger not allowed
-104	RECOVERABLE	Data type
-101	RECOVERABLE	Invalid character
-100	RECOVERABLE	Command error
0	NO_SEVERITY	Queue is empty
603	RECOVERABLE	Power on state lost
605	RECOVERABLE	Calibration dates lost
820	RECOVERABLE	Parsing value
900	FATAL	Internal system
1100	RECOVERABLE	Command unavailable
1101	RECOVERABLE	Parameter too big
1102	RECOVERABLE	Parameter too small
1103	RECOVERABLE	Min greater than max
1104	RECOVERABLE	Too many digits for param type

Error and status messages

Error number	Error level	Error message
1107	RECOVERABLE	Cannot modify factory menu
1108	RECOVERABLE	Menu name does not exist
1109	RECOVERABLE	Menu name already exists
1112	RECOVERABLE	Password entered does not match current password
1114	RECOVERABLE	Settings conflict with %s, where %s represents specifics on what the conflict is
1115	RECOVERABLE	Parameter error %s, where %s explains why parameter error
1116	RECOVERABLE	Configuration error %s, where %s explains why configuration error
1200	RECOVERABLE	TSP-Link initialization failed
1201	RECOVERABLE	TSP-Link initialization failed
1202	RECOVERABLE	TSP-Link initialization failed
1203	RECOVERABLE	TSP-Link initialization failed (possible loop in node chain)
1204	RECOVERABLE	TSP-Link initialization failed
1205	RECOVERABLE	TSP-Link initialization failed (no remote nodes found)
1206	RECOVERABLE	TSP-Link initialization failed
1207	RECOVERABLE	TSP-Link initialization failed
1208	RECOVERABLE	TSP-Link initialization failed
1209	RECOVERABLE	TSP-Link initialization failed
1210	RECOVERABLE	TSP-Link initialization failed (node ID conflict)
1211	RECOVERABLE	Node %u is inaccessible, where %u represents a number
1212	RECOVERABLE	Invalid node ID
1213	RECOVERABLE	TSP-Link session expired
1214	RECOVERABLE	TSP-Link unknown remote command encoding
1215	RECOVERABLE	Code execution requested within the local group
1216	RECOVERABLE	Remote execution requested on node in group with pending overlapped operations
1217	RECOVERABLE	Remote execution requested on node outside the local group
1400	RECOVERABLE	Expected at least %d parameters, where %d represents a number
1401	RECOVERABLE	Parameter %d is invalid, where %d represents a number
1402	RECOVERABLE	User scripts lost
1403	RECOVERABLE	Factory scripts lost
1404	RECOVERABLE	Invalid byte order
1405	RECOVERABLE	Invalid ASCII precision
1406	RECOVERABLE	Invalid data format
1600	RECOVERABLE	Maximum GPIB message length exceeded
1601	RECOVERABLE	GPIB input queue overrun
1800	RECOVERABLE	Invalid digital trigger mode
1801	RECOVERABLE	Invalid digital I/O line
1802	RECOVERABLE	Digital bit in parameter write protected
2100	FATAL	Could not open socket
2101	FATAL	Could not close socket
2102	RECOVERABLE	LAN configuration already in progress
2103	RECOVERABLE	LAN disabled
2104	RECOVERABLE	Socket error
2105	RECOVERABLE	Unreachable gateway

Error and status messages

Error number	Error level	Error message
2106	RECOVERABLE	Could not acquire ip address
2107	RECOVERABLE	Duplicate IP address detected
2108	RECOVERABLE	DHCP lease lost
2109	RECOVERABLE	LAN cable disconnected
2110	RECOVERABLE	Could not resolve hostname
2111	RECOVERABLE	DNS name (FQDN) too long
2112	RECOVERABLE	Connection not established
2200	RECOVERABLE	File write error
2201	RECOVERABLE	File read error
2202	RECOVERABLE	Cannot close file
2203	RECOVERABLE	Cannot open file
2204	RECOVERABLE	Directory not found
2205	RECOVERABLE	File not found
2206	RECOVERABLE	Cannot read current working directory
2207	RECOVERABLE	Cannot change directory
2208	RECOVERABLE	Cannot create directory
2209	RECOVERABLE	Cannot remove directory
2210	RECOVERABLE	File is not a valid script format
2211	RECOVERABLE	File system error
2212	RECOVERABLE	File system command not supported
2213	RECOVERABLE	Too many open files
2214	RECOVERABLE	File access denied
2215	RECOVERABLE	Invalid file handle
2216	RECOVERABLE	Invalid drive
2217	RECOVERABLE	File system busy
2218	RECOVERABLE	Disk full
2219	RECOVERABLE	File corrupt
2220	RECOVERABLE	File already exists
2221	RECOVERABLE	File seek error
2222	RECOVERABLE	End-of-file error
2223	RECOVERABLE	Directory not empty
2300	RECOVERABLE	Upgrade found not upgradable
2301	RECOVERABLE	Upgrade uncompress failed
2302	RECOVERABLE	Upgrade device not ready
2303	RECOVERABLE	Upgrade device type not acceptable
2304	RECOVERABLE	Upgrade write to device checksum failure
2305	RECOVERABLE	Upgrade write to device failed
2306	RECOVERABLE	Upgrade timeout connect with device
2307	RECOVERABLE	Upgrade failure
2400	RECOVERABLE	Invalid specified connection
2401	RECOVERABLE	Invalid timeout seconds (.001 to 30)
2402	RECOVERABLE	TSPnet remote error: %s, where %s explains the remote error
2403	RECOVERABLE	TSPnet failure
2404	RECOVERABLE	TSPnet read failure
2405	RECOVERABLE	TSPnet read failure, aborted

Error and status messages

Error number	Error level	Error message
2406	RECOVERABLE	TSPnet read failure, timeout
2407	RECOVERABLE	TSPnet write failure
2408	RECOVERABLE	TSPnet write failure, aborted
2409	RECOVERABLE	TSPnet write failure, timeout
2410	RECOVERABLE	TSPnet max connections reached
2411	RECOVERABLE	TSPnet connection failed
2412	RECOVERABLE	TSPnet invalid termination
2413	RECOVERABLE	TSPnet invalid reading buffer table
2414	RECOVERABLE	TSPnet invalid reading buffer index range
2415	RECOVERABLE	TSPnet feature only supported on TSP connections
2416	RECOVERABLE	TSPnet musty specify both port and init
2417	RECOVERABLE	TSPnet disconnected by other side
4900	RECOVERABLE	Reading buffer index %d is invalid, where %d represents a number
4901	RECOVERABLE	The maximum index for this buffer is %d, where %d represents a number
4902	RECOVERABLE	Reading buffers must be able to contain at least one element
4903	RECOVERABLE	Reading buffer expired
4904	RECOVERABLE	ICX parameter count mismatch, %s (Line #%d), where %s and %d provide more information on error
4905	RECOVERABLE	ICX parameter invalid value, %s (Line #%d), where %s and %d provide more information on error
4906	RECOVERABLE	ICX invalid function id, %s (Line #%d), where %s and %d provide more information on error
4907	RECOVERABLE	Cannot modify built-in reading buffers
4908	RECOVERABLE	Cannot change this setting unless buffer is cleared
4909	RECOVERABLE	Reading buffer not found within device
4910	RECOVERABLE	No readings exist within buffer
4911	RECOVERABLE	Table not found within buffer
4912	RECOVERABLE	Attribute not found within buffer
4914	RECOVERABLE	Index exceeds maximum readings stored in buffer
4915	RECOVERABLE	Attempting to store past capacity of reading buffer
5500	RECOVERABLE	Card unknown error
5501	RECOVERABLE	Failed card NVMEM write
5502	RECOVERABLE	Failed card NVMEM read
5503	RECOVERABLE	Closure count lost
5504	RECOVERABLE	Temperature sensor failure
5505	RECOVERABLE	Error completing a card action in requested operation
5506	RECOVERABLE	Communication error with a card in requested operation
5507	RECOVERABLE	Card operation completed under low total power
5508	RECOVERABLE	Card operation completed under low bank power
5509	RECOVERABLE	Card operation completed under low slot power
5510	RECOVERABLE	Not enough total power to hold requested card operation
5511	RECOVERABLE	Not enough bank power to hold requested card operation
5512	RECOVERABLE	Not enough slot power to hold requested card operation
5513	RECOVERABLE	Not enough total power to complete requested card operation
5514	RECOVERABLE	Not enough bank power to complete requested card operation

Error and status messages

Error number	Error level	Error message
5515	RECOVERABLE	Not enough slot power to complete requested card operation
5516	RECOVERABLE	Slot empty, no configuration data exist
5517	RECOVERABLE	Slot error, configuration data not found
5518	RECOVERABLE	Slot error, communication error accessing configuration data
5519	RECOVERABLE	Slot error, timeout error accessing configuration data
5520	RECOVERABLE	Channel error, channel list contains a channel not in system
5521	RECOVERABLE	Parameters adjusted, must recreate scan
5522	RECOVERABLE	Scan running, must abort scan
5600	RECOVERABLE	10 vdc zero error
5601	RECOVERABLE	100 vdc zero error
5602	RECOVERABLE	10 vdc full scale error
5603	RECOVERABLE	-10 vdc full scale error
5604	RECOVERABLE	100 vdc full scale error
5605	RECOVERABLE	100m vdc zero error
5606	RECOVERABLE	100 2-w zero error
5607	RECOVERABLE	10k 2-w zero error
5608	RECOVERABLE	100k 2-w zero error
5609	RECOVERABLE	10M 2-w zero error
5610	RECOVERABLE	10M 2-w full scale error
5611	RECOVERABLE	10M 2-w open error
5612	RECOVERABLE	100 4-w zero error
5613	RECOVERABLE	10k 4-w zero error
5614	RECOVERABLE	100k 4-w zero error
5615	RECOVERABLE	10M 4-w sense lo zero error
5616	RECOVERABLE	1k 4-w full scale error
5617	RECOVERABLE	10k 4-w full scale error
5618	RECOVERABLE	100k 4-w full scale error
5619	RECOVERABLE	1M 4-w full scale error
5620	RECOVERABLE	10M 4-w full scale error
5621	RECOVERABLE	10m adc zero error
5622	RECOVERABLE	100m adc zero error
5623	RECOVERABLE	10m adc full scale error
5624	RECOVERABLE	100m adc full scale error
5625	RECOVERABLE	1 adc full scale error
5626	RECOVERABLE	2k 4-w dckt Ioff zero error
5627	RECOVERABLE	2k 4-w dckt Ion zero error
5628	RECOVERABLE	1k 4-w dckt Ioff zero error
5629	RECOVERABLE	1k 4-w dckt Ion zero error
5630	RECOVERABLE	100 4-w dckt Ioff zero error
5631	RECOVERABLE	100 4-w dckt Ion zero error
5632	RECOVERABLE	10 4-w dckt Ioff zero error
5633	RECOVERABLE	10 4-w dckt Ion zero error
5634	RECOVERABLE	1 4-w dckt Ion zero error
5635	RECOVERABLE	10 2-w zero error
5636	RECOVERABLE	10 4-w full scale error
5637	RECOVERABLE	100 4-w full scale error

Error and status messages

Error number	Error level	Error message
5638	RECOVERABLE	10u adc zero error
5639	RECOVERABLE	100u adc zero error
5640	RECOVERABLE	1m adc zero error
5641	RECOVERABLE	1 adc zero error
5642	RECOVERABLE	10u adc full scale error
5643	RECOVERABLE	100u adc full scale error
5644	RECOVERABLE	1m adc full scale error
5645	RECOVERABLE	1 vac fast noise error
5646	RECOVERABLE	1 vac fast full scale error
5647	RECOVERABLE	100m vac dac error
5648	RECOVERABLE	1 vac dac error
5649	RECOVERABLE	10 vac dac error
5650	RECOVERABLE	100 vac dac error
5651	RECOVERABLE	100m vac zero error
5652	RECOVERABLE	100m vac full scale error
5653	RECOVERABLE	1 vac zero error
5654	RECOVERABLE	1 vac full scale error
5655	RECOVERABLE	1 vac noise error
5656	RECOVERABLE	10 vac zero error
5657	RECOVERABLE	10 vac full scale error
5658	RECOVERABLE	10 vac noise error
5659	RECOVERABLE	100 vac zero error
5660	RECOVERABLE	100 vac full scale error
5661	RECOVERABLE	300 vac zero error
5662	RECOVERABLE	300 vac full scale error
5663	RECOVERABLE	300 vac noise error
5664	RECOVERABLE	Post filter offset error
5665	RECOVERABLE	1 aac zero error
5666	RECOVERABLE	1 aac full scale error
5667	RECOVERABLE	3 aac zero error
5668	RECOVERABLE	3 aac full scale error
5669	RECOVERABLE	1V 10 Hz amplitude error
5670	RECOVERABLE	Frequency gain error
5671	RECOVERABLE	100 Ohm Ioff Ocomp FS error
5672	RECOVERABLE	10k Ohm Ioff Ocomp FS error
5673	RECOVERABLE	Temperature cold cal error
5674	RECOVERABLE	Analog output zero error
5675	RECOVERABLE	Analog output pos. gain error
5676	RECOVERABLE	Analog output neg. gain error
5677	RECOVERABLE	100 4-w dckt Ioff full scale error
5678	RECOVERABLE	100 4-w dckt Ion full scale error
5679	RECOVERABLE	10 4-w dckt full scale error
5680	RECOVERABLE	1 4-w dckt Ion full scale error
5681	RECOVERABLE	10k 4-w ocomp Ioff full scale error
5682	RECOVERABLE	10k 4-w ocomp Ion full scale error
5683	RECOVERABLE	2k 4-w dckt Ioff full scale error

Error and status messages

Error number	Error level	Error message
5684	RECOVERABLE	2k 4-w dckt Ion full scale error
5685	RECOVERABLE	1k 4-w dckt Ioff full scale error
5686	RECOVERABLE	1k 4-w dckt Ion full scale error
5687	RECOVERABLE	10 4-w zero error
5688	RECOVERABLE	10 4-w Ioff zero error
5689	RECOVERABLE	1m aac full scale error
5690	RECOVERABLE	1m aac zero error
5691	RECOVERABLE	10m aac full scale error
5692	RECOVERABLE	10m aac zero error
5693	RECOVERABLE	100m aac full scale error
5694	RECOVERABLE	100m aac zero error
5695	RECOVERABLE	Offset calibration error
5696	RECOVERABLE	1V 10 Hz frequency error
5697	RECOVERABLE	Calibration data invalid
5698	RECOVERABLE	AC calibration data lost
5699	RECOVERABLE	DC calibration data lost
5700	RECOVERABLE	PreCal calibration data lost
5701	RECOVERABLE	A/D timeout
5702	RECOVERABLE	1 4-w dckt Ioff zero error
5703	RECOVERABLE	100 4-w Ioff zero error
5704	RECOVERABLE	10k 4-w Ioff zero error
5705	RECOVERABLE	10 4-w dckt Ioff full scale error
5706	RECOVERABLE	1 4-w dckt Ioff full scale error
5707	RECOVERABLE	1k TRTD HI Ion zero error
5708	RECOVERABLE	1k TRTD HI Ioff zero error
5709	RECOVERABLE	1k TRTD SLO Ion zero error
5710	RECOVERABLE	1k TRTD SLO Ioff zero error
5711	RECOVERABLE	10k TRTD HI Ion zero error
5712	RECOVERABLE	10k TRTD HI Ioff zero error
5713	RECOVERABLE	10k TRTD SLO Ion zero error
5714	RECOVERABLE	10k TRTD SLO Ioff zero error
5715	RECOVERABLE	100k TRTD HI Ion zero error
5716	RECOVERABLE	100k TRTD SLO Ion zero error
5717	RECOVERABLE	1k TRTD HI Ion full scale error
5718	RECOVERABLE	1k TRTD HI Ioff full scale error
5719	RECOVERABLE	1k TRTD SLO Ion full scale error
5720	RECOVERABLE	1k TRTD SLO Ioff full scale error
5721	RECOVERABLE	10k TRTD HI Ion full scale error
5722	RECOVERABLE	10k TRTD HI Ioff full scale error
5723	RECOVERABLE	10k TRTD SLO Ion full scale error
5724	RECOVERABLE	10k TRTD SLO Ioff full scale error
5725	RECOVERABLE	100k TRTD HI Ion full scale error
5726	RECOVERABLE	100k TRTD SLO Ion full scale error
5727	RECOVERABLE	10 vdc full scale 6p4 error
5728	RECOVERABLE	10 vdc full scale p64 error
5729	RECOVERABLE	10 vdc zero 6p4 error

Error and status messages

Error number	Error level	Error message
5730	RECOVERABLE	10 vdc zero p64 error
5731	RECOVERABLE	1k 4-w ocomp Ioff full scale error
5732	RECOVERABLE	Questionable calibration
5733	RECOVERABLE	Questionable temperature
5734	RECOVERABLE	Internal DMM system error
5735	RECOVERABLE	General unknown DMM error
5736	RECOVERABLE	Untranslated DMM error
5737	RECOVERABLE	Error completing DMM action in requested operation
5738	RECOVERABLE	Communication error with DMM in requested operation
5739	RECOVERABLE	DMM calibration error occurred during processing command
5740	RECOVERABLE	DMM calibration error occurred setting adjustment date
5741	RECOVERABLE	DMM calibration error occurred getting adjustment date
5742	RECOVERABLE	DMM calibration error occurred setting verify date
5743	RECOVERABLE	DMM calibration error occurred getting verify date
5744	RECOVERABLE	DMM calibration error occurred setting password
5745	RECOVERABLE	DMM calibration error occurred getting password
5746	RECOVERABLE	DMM calibration error occurred setting count
5747	RECOVERABLE	DMM calibration error occurred getting count

Frequently asked questions (FAQs)

In this section:

How do I get my LAN or web connection to work?

For troubleshooting suggestions, see Troubleshooting LAN interfaces (on page 9-5).

For more detailed information on remote interface connections, see Communications interfaces (see "Remote communication interfaces" on page 2-53).

Why can't I close a channel?

The channel might be set to be forbidden to close.

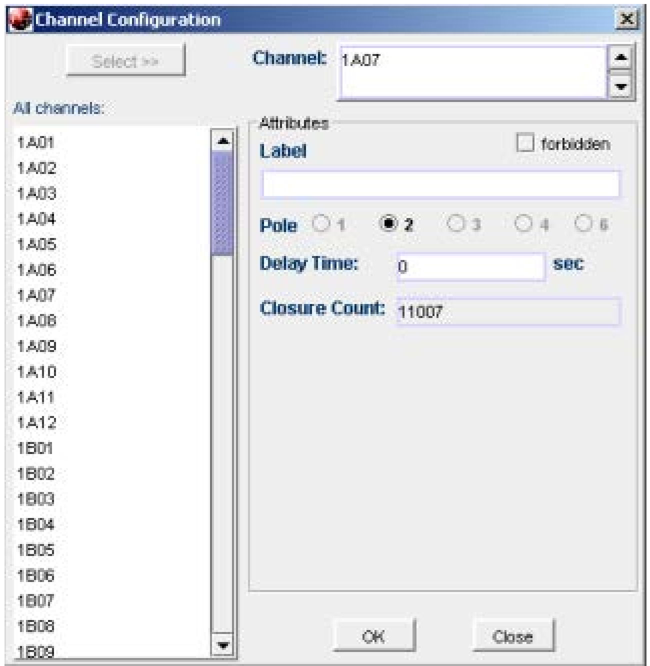
To check the forbidden state of a channel from the front panel:

1. Display a channel (you might need to press **DISPLAY**).
2. Use the navigation wheel to select the channel you want to check.
3. Press **CONFIG**, then press **CHAN**.
4. Select **FORBID**.
5. Press **ENTER**.
6. **Yes** and **No** are displayed. The current selection blinks. To change the setting to allow the channel to close, select **No**.

To check the forbidden state of a channel from the web interface:

1. From the list on the left, select the slot that contains the channel.
2. Right-click the channel. The Channel Configuration dialog box is displayed.

Figure 125: Channel configuration dialog box



3. If the forbidden box is selected, the channel is forbidden to close. To allow the channel to close, clear the box.
4. Click **OK** to save the change.

To check the forbidden state of a channel from a remote interface:

You can also clear, check, and set the forbidden state of channels using the following commands:

- channel.clearforbidden() (on page 8-49)
- channel.getforbidden() (on page 8-66)
- channel.setforbidden() (on page 8-94)

How do I know if an error has occurred on my instrument?

If you are using TSB Embedded, error messages are displayed in the Instrument Output box when they occur.

If you are using another remote interface, you might need to use commands to retrieve the error messages. You can use the commands errorqueue.count (on page 8-246) and errorqueue.next() (on page 8-246) to retrieve the number of messages and the text of the messages.

To set the instrument to automatically send generated errors, set localnode.showerrors (on page 8-300) to 1 (enabled).

To set the instrument to automatically send prompts after each command message, set localnode.prompts (on page 8-296) to 1 (enabled).

How do I find the serial number and firmware version of the instrument?

The serial number is on the rear panel of the instrument. You can also use the front panel **MENU** option to display the serial number and firmware version.

To display serial number or firmware revision on the front panel:

1. If the Series 3700A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Use the navigation wheel ◷ to scroll to the **UNIT-INFO** menu.
4. Press the **ENTER** key.

On the UNIT INFORMATION menu, scroll to the **SERIAL#** or **FIRMWARE** option and press the **ENTER** key. The Series 3700A serial number is displayed.

Section 11

Next steps

In this section:

Additional Series 3700A information

For additional information about the Series 3700A, refer to:

- The Product Information CD-ROM (ships with the product): Contains software tools, drivers, and product documentation, including documentation for switch cards that are compatible with the Series 3700A
- The Keithley Instruments website (*http://www.keithley.com*): Contains the most up-to-date information. From the website, you can access:
  - The Knowledge Center, which contains the following handbooks:
    - *The Low Level Measurements Handbook: Precision DC Current, Voltage, and Resistance Measurements*
  - Application notes
  - Updated drivers
  - Information about related products, including:
    - The Model 4200-SCS Semiconductor Characterization System
- Your local Field Applications Engineer can help you with product selection, configuration, and usage. Check the website for contact information.

Appendix A

Maintenance

In this appendix:

Introduction

This section provides maintenance information and procedures that can be performed by the operator.

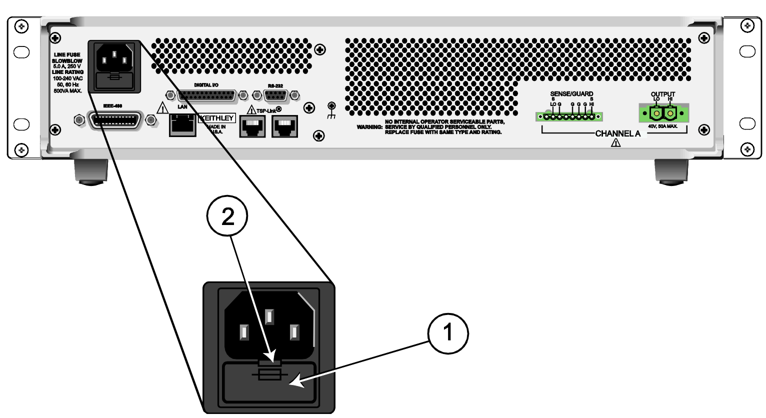
Line fuse replacement

A fuse located on the Series 3700A rear panel protects the power line input of the instrument.

WARNING

Disconnect the line cord at the rear panel and remove all test leads connected to the instrument before replacing the line fuse. Failure to do so could expose the operator to hazardous voltages that could result in personal injury or death.

Figure 126: Fuse replacement



⚠ WARNING

To prevent injury, death, or instrument damage, use only the correct fuse type (see table).

Perform the following steps to replace the line fuse:

1. Power off the instrument and remove the line cord.
2. The fuse drawer (item 1 in the figure) is located below the AC receptacle. A small tab is located on the top of the fuse drawer (item 2). Using a thin-bladed knife or a screwdriver, pry this tab away from the AC receptacle.
3. Slide the fuse drawer out to gain access to the fuse (the fuse drawer does not pull completely out of the power module).
4. Snap the fuse out of the drawer and replace it with the same type (the fuse is specified in the table below).
5. Push the fuse drawer back into the module.

If the power line fuse continues to blow, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

Fuse replacement

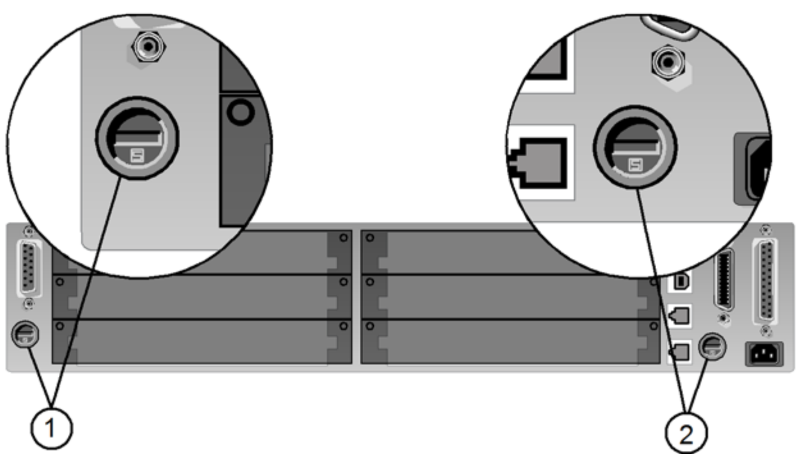
The analog backplane AMPS fuse (see item 1 in Fuse location figure) is accessible from the rear panel, just below the analog backplane connector. The instrument fuse (see item 2 in Fuse location figure) is accessible from the rear panel, below the GPIB Connector.

⚠ WARNING

Disconnect all external power from the equipment and the line cord before performing any maintenance on the Model 3706A.

Failure to disconnect all power may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

Figure 127: Fuse location



Fuse location	Rating	Keithley Instruments part number
(1) Analog backplane fuse	250V, 3A fast blow 5x20mm	FU-99-1
(2) Instrument fuse	250V / 1.25A slow blow 5x20mm	FU-106-1.25

To replace a fuse:

1. Using a flat-tip screwdriver, disengage the fuse holder by rotating it counter-clockwise.
2. Pull out the fuse holder and replace the fuse with the correct type (see table).
3. Reinstall the fuse holder.

If the fuse continues to blow, a circuit malfunction exists and must be corrected. Return the instrument to Keithley Instruments for repair.

AMPS analog backplane fuse replacement

⚠ WARNING

Make sure the instrument is disconnected from the power line and other equipment before replacing the AMPS fuse.

⚠ CAUTION

Do not use a fuse with a higher current rating than specified or instrument damage may occur. If the instrument repeatedly blows fuses, locate and correct the cause of the trouble before replacing the fuse.

NOTE

Model 3721 card supports both AC and DC current measurements. Refer to the Schematic (see "Rear panel, backplane, and DMM connect relays schematic" on page 5-1) in the Series 3700A Switching and Control Cards Reference Manual. The Model 3721 card has replaceable fuses. For replacement information, refer to the "Model 3721: AMPS channels fuse replacement" section in the Series 3700A Switching and Control Cards Reference Manual.

1. Turn off the power and disconnect the power line and connections.
2. From the rear panel, gently push in the AMPS fuse holder with a flat blade screwdriver and rotate the fuse holder one-quarter turn counterclockwise.
3. Remove the fuse and replace it with the same type (3A, 250V, fast-blow, 5 × 20mm). The Keithley Instruments part number is FU-99-1.
4. Install the new fuse by reversing the procedure above.

Front panel tests

You can test the functionality of the front panel keys and the display.

Test procedure

This procedure tests the functionality of each front panel key and the display.

To run the test:

1. If the Series 3700A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Display the MAIN MENU by pressing the **MENU** key.
3. Turn the navigation wheel to scroll to the **DISPLAY** menu item. Press the **ENTER** key to select.
4. Press the **ENTER** key to select **TEST**.
5. Turn the navigation wheel until the **KEYS** menu item is highlighted.
6. To start the test, press the **ENTER** key.
7. Press a key. The label name for that key is displayed, which indicates that it is functioning properly. When the key is released, the message “No keys pressed” is displayed.
8. When the test is complete, press the **EXIT (LOCAL)** key twice The FRONT PANEL TESTS menu is displayed.
9. Select **DISPLAY-PATTERNS**. This test lets you verify that each pixel and indicator in the vacuum fluorescent display is working properly.
10. To start the test, press the **ENTER** key.
11. The checkerboard pattern and the annunciators that are on during normal operation are displayed. Verify that they are displayed correctly.
12. Press the **ENTER** key.
13. The checkerboard pattern (alternate pixels on) and all annunciators are displayed. Verify that they are displayed correctly.
14. Press the **ENTER** key.
15. Each digit (and adjacent annunciator) is sequenced. All of the pixels of the selected digit are on. Verify that they are displayed correctly.
16. Press the **EXIT** key to end the test.
17. Continue pressing the **EXIT** key to back out of the menu structure.

Keys test

This test lets you check the functionality of each front panel key.

Perform the following steps to run the KEYS test:

1. If the Series 3700A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Navigate through the menus by turning the navigation wheel. Press the **ENTER** key to select the menu items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel until the **KEYS** menu item is highlighted.
5. To start the test, press the **ENTER** key. While the test is active, when you press a key, the label name for that key is displayed to indicate that it is functioning properly. When you release the key, the message “No keys pressed” is displayed.
6. To test the EXIT (LOCAL) key, press the **EXIT (LOCAL)** key once.
7. To exit the test, press the **EXIT (LOCAL)** key twice consecutively. You will exit the test and the instrument returns to the FRONT PANEL TESTS menu.
8. Press the **EXIT (LOCAL)** key multiple times to exit out of the menu structure.

Display patterns test

This test lets you verify that each pixel and indicator in the vacuum fluorescent display is working properly.

Perform the following steps to run the display test:

1. If the Series 3700A is in remote mode, press the **EXIT (LOCAL)** key once to place the instrument in local mode.
2. Press the **MENU** key.
3. Navigate through the menus by turning the navigation wheel, and then pressing the **ENTER** key to select the items as follows: **DISPLAY > TEST > DISPLAY-TESTS**.
4. Turn the navigation wheel until the **DISPLAY-PATTERNS** menu item is highlighted.
5. To start the display test, press the **ENTER** key. There are three parts to the display test. Each time the **ENTER** key or the navigation wheel is pressed, the next part of the test sequence is selected. The three parts of the test sequence are as follows:
   - Checkerboard pattern and the indicators that are on during normal operation
   - Checkerboard pattern (alternate pixels on) and all the numeric indicators (which are not used) are illuminated
   - Each digit (and adjacent indicators) is sequenced; all of the pixels of the selected digit are on

1. When finished, abort the display test by pressing the **EXIT (LOCAL)** key. The instrument returns to the FRONT PANEL TESTS menu. Continue pressing the **EXIT (LOCAL)** key to exit out of the menu structure.

Displaying the instrument's serial number

The instrument serial number is on a label on the rear panel of the instrument. You can also access the serial number from the front panel using the front-panel keys and menus.

To display the serial number on the front panel:

1. If the Series 3700A is in remote operation, press the **EXIT (LOCAL)** key once to place the instrument in local operation.
2. Press the **MENU** key.
3. Use the navigation wheel ◔ to scroll to the **UNIT-INFOSYSTEM-INFO** menu item.
4. Press the **ENTER** key. The SYSTEM INFORMATION menu is displayed.
5. Scroll to the **SERIAL#** menu item.
6. Press the **ENTER** key. The Series 3700A serial number is displayed.

Upgrading the firmware

Use this procedure to upgrade the Model 3706A firmware directly from a USB flash drive using a file. The upgrade process should take approximately five minutes, depending on the cards in the system and if a digital multimeter (DMM) is installed.

The normal upgrade procedure only upgrades to a higher level software version. If any part of the system is already at a higher software revision, that part of the system is skipped during the upgrade. A separate operation is available to revert to an earlier revision firmware.

The upgrade process upgrades not only the mainframe, but also the DMM and any cards in the system. Make sure all available cards are populated in the mainframe before beginning the upgrade procedure.

You can upgrade the firmware using a USB flash drive on the front panel, through the web interface, or using remote interface.

NOTE

You can upgrade a single card at a later time by installing the card in the instrument and re-running the upgrade procedure. The upgrade procedure will verify that the instrument firmware is at the latest revision and will only upgrade the additional installed card.

Upgrade files are available on the Keithley Instruments website (*http://www.keithley.com*).

To locate the upgrade files on the Keithley website:

1. Select the **Support** tab.
2. Search for your model number's firmware:
    a. In the model number box, type **3700A**.
    b. Select **Firmware**.
    c. Click the search button.
3. A list of available firmware updates and any available documentation for the instrument is displayed. Click the desired file to download.

CAUTION

Disconnect the input and output terminals before you upgrade.

Do not remove power from the Series 3700A or remove the flash drive while an upgrade is in progress. Wait until the instrument completes the upgrade procedure and the opening display is shown.

Upgrade procedure using the remote interface

You can also upgrade or revert to a previous version of firmware using the remote interface command upgrade.unit() (on page 8-463) or upgrade.previous() (on page 8-462).

NOTE

For models without a front panel, the LAN Status and clock status LEDs blink in unison during the upgrade process.

Firmware upgrade from a USB flash drive

Use this procedure to upgrade the Model 3706A firmware directly from a USB flash drive using a *.cab file. The upgrade process should take approximately 5 minutes, depending on the cards in the system and if a DMM is installed.

The normal upgrade procedure only upgrades to a higher level software version. If any part of the system is already at a higher software revision, that part of the system is skipped during the upgrade. A separate operation is available to upgrade to earlier revision firmware.

The upgrade process upgrades not only the mainframe, but also the digital multimeter (DMM) and any cards in the system. Make sure all available cards are populated in the mainframe before beginning the upgrade procedure.

Firmware upgrade procedure using the front panel USB port

NOTE

You can upgrade a single card at a later time by installing the card in the instrument and re-running the upgrade procedure. The upgrade procedure will verify that the instrument firmware is at the latest revision and will only upgrade the additional installed card.

CAUTION

Do not turn off the instrument or remove the flash drive during the upgrade procedure. Wait until the instrument completes the upgrade procedure and the opening display is shown.

NOTE

If your model does not have a front panel, upgrade over the remote interface with the appropriate command (upgrade.unit() (on page 8-463) or upgrade.previous() (on page 8-462)).

To upgrade the firmware using the front panel USB port:

1. Copy upgrade *.cab file to a blank USB flash drive. Ensure that the drive size is large enough for the size of the upgrade file.

NOTE

Verify that the USB flash drive is blank.

2. Power on the instrument
3. Install a USB flash drive in the front panel connector
4. On the front panel, press the **MENU** key
5. Turn the navigation wheel to scroll to **UPGRADE** and press the navigation wheel.
6. The question **UPGRADE UNIT?** is displayed. Select **Previous** to install a previous version or select **Yes** to upgrade to a newer version and press the navigation wheel.
7. The upgrade status is displayed on the front panel, including the percentage complete. When the file has been unpacked, the upgrade status will be displayed as it is upgraded (first cards installed in the slots including the DMM if installed, and then the Main Series 3700).

NOTE

For models without a front panel, the LAN Status and clock status LEDs blink in unison during the upgrade process.

The instrument will reboot automatically when the upgrade is complete.

To upgrade the firmware from the web interface:

1. Access the instrument's web page (for additional information, see Step 5: Access the instrument's web page (on page B-9)).
2. From the left navigation area, select **Unit**.
3. Log in if necessary.
4. From the Unit buttons, click **Upgrade Firmware**.A confirmation message is displayed.
5. A version message is displayed. Select the appropriate option.Select the file that contains the appropriate version of firmware.
6. Click **Open**. A progress dialog box is displayed. When the upgrade begins, the front panel display will also display the progress.

   During the upgrade, you will see messages that indicate that the connection has been lost. This is normal.
7. After the instrument automatically restarts, it will be ready for use.

Appendix B

LAN concepts and settings

In this appendix:

Overview

The Keithley Instruments Series 3700A System Switch/Multimeter is LXI version 1.4 Core 2011 compliant. The Series 3700A is a scalable test system that can connect directly to a host computer or interact with a DHCP or DNS server and other LXI-compliant instruments on a local area network (LAN). The Model 3706A also supports Multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), which are useful on a LAN with no central administration.

The Series 3700A is compliant with the IEEE Std 802.3 and supports full connectivity on a 10 or 100 megabits-per-second network. The LAN interface is an alternative to GPIB that can be used to build flexible test systems that include web access.

NOTE

Please read this entire section before you connect the Series 3700A to the LAN.

Establishing a point-to-point connection

To enable access to the instrument web page and other web applications from a computer, use a one-to-one LAN connection and set up a static IP address between the host computer and the instrument.

The following instructions describe how to configure the instrument's IP address. The instrument's IP address is based on the present IP address of the host computer. Each device on the LAN (corporate or private) requires a unique IP address.

CAUTION

Contact your corporate information technology (IT) department for permission before you connect the Series 3700A to a corporate network.

If you have problems, see LAN troubleshooting suggestions (see "Verify connections and settings" on page 9-6).

NOTE

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 Enabled = NO (disabled).

Be sure to return all settings to their original configuration before reconnecting the host computer to a corporate network. Failure to do this could result in loss of data. Contact your system administrator for more information.

Step 1: Identify and record the existing IP configuration

To identify the existing IP configuration:

1. Open a command prompt window:

   Microsoft® Windows® 2000 or Windows XP:

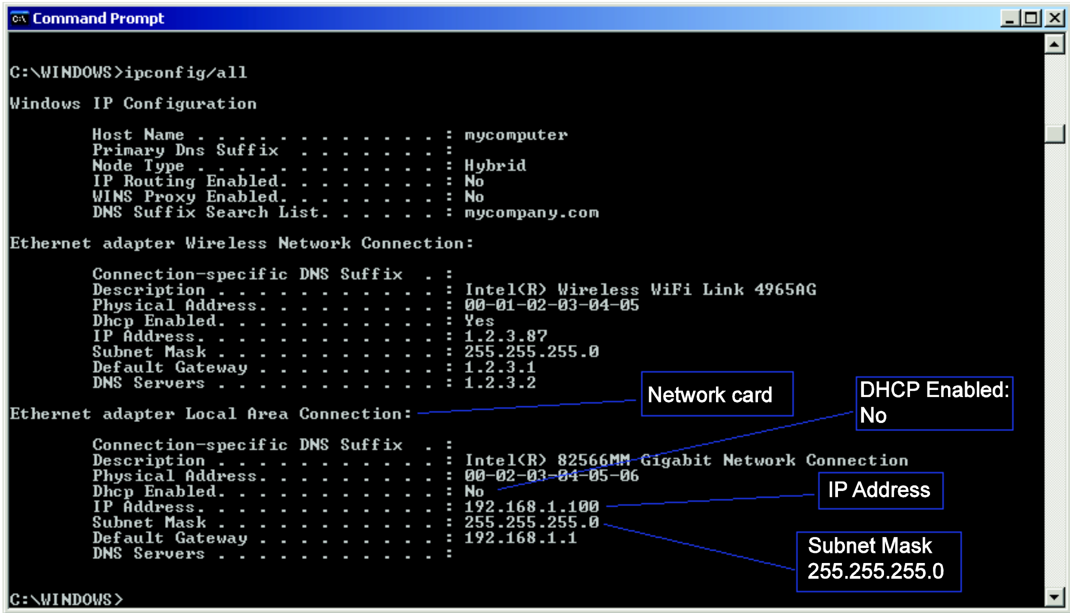
   a. Click **Start** and select **Run**.
   b. In the Open field, type `cmd`.
   c. Click **OK**.

   Microsoft Windows Vista® or Windows 7:

   a. Click **Start**.
   b. Select **All Programs > Accessories > Command Prompt**.

2. At the command prompt, type `ipconfig/all` and press the **Enter** key. A list of existing IP configuration information for your computer is displayed.

Figure 128: Computer IP configuration using the command prompt



NOTE

If the information for the ethernet adapter displays "`Media Disconnected`," close the command prompt and go to Step 2: Disable DHCP to use the computer's existing IP address (on page B-4).

3. When the information is displayed, record the following information for the network card:
   - DHCP mode: ________________________________
   - IP address: _________________________________
   - Subnet mask: _______________________________
   - Default gateway: _____________________________
   - DNS servers: ________________________________

⚠ CAUTION

The `ipconfig/all` command displays the configuration of every network card. Make sure that you record the information for the proper network card.

1. If:
   - **DHCP Enabled = Yes:** Go to Step 2: Disable DHCP to use the computer's existing IP address (on page B-4)
   - **DHCP Enabled = No:** Go to Step 3: Configure the instrument's LAN settings (on page B-8).
1. To exit the IP configuration screen, type **exit** at the command prompt and press **Enter**.

Step 2: Disable DHCP to use the computer's existing IP address

NOTE

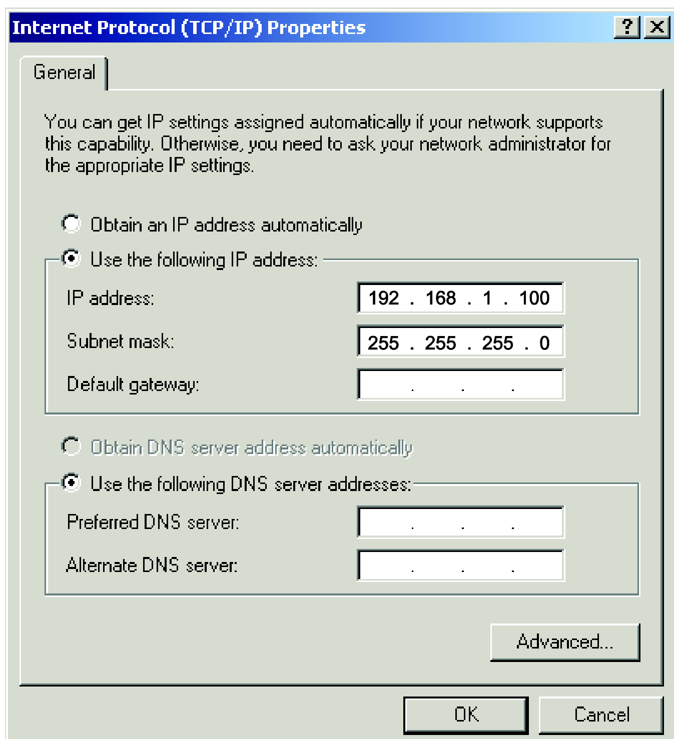
Do not change the IP address at any time without talking to your system administrator. Entering an incorrect IP address can prevent your workstation from connecting to your corporate network.

See the appropriate instructions below for your operating system. These instructions show the default options. Be aware that there may be differences in these steps if your Microsoft Windows options are customized or if you do not have administrator status.

Windows 2000: To disable DHCP:

1. Click **Start > Settings > Control Panel**.
2. Open **Network and Dial-up connections**.
3. Right-click **Local Area Connection** and select **Properties**. The Local Area Connection Properties dialog box is displayed.
4. Double-click **Internet Protocol (TCP/IP)** in the items list. The Internet Protocol (TCP/IP) Properties dialog box is displayed, as shown here.

Figure 129: Internet Protocol (TCP/IP) Properties dialog box

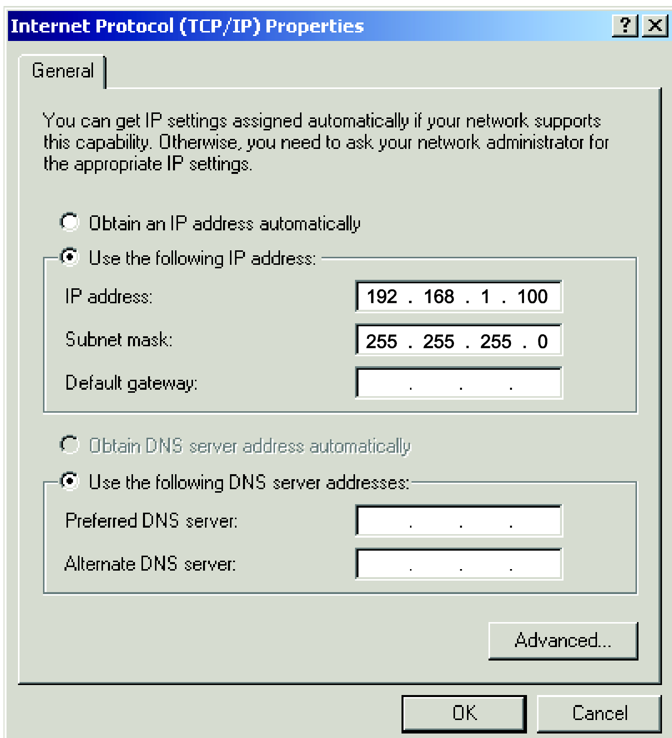


5. Select **Use the following IP address**. The option for "Use the following DNS server addresses" is automatically selected.
6. Set the IP address. If the IP address and subnet mask fields:
   - **Contain values:** Record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the instrument's LAN settings (on page B-8).
   - **Are blank:** In the IP address field, enter 192.168.1.100. In the subnet mask field, enter 255.255.255.0. These will be used to configure the LAN settings of the instrument.
1. Click **OK** to close the Internet Protocol (TCP/IP) Properties dialog box.
2. Click **OK** to close the Local Area Connection Properties dialog box.
3. Close the Network Connections window.

Windows XP: To disable DHCP:

1. Click **Start > Settings > Control Panel**.
2. Open **Network Connections**.
3. Right-click **Local Area Connection** and select **Properties**. The Local Area Connection Properties dialog box is displayed.
4. In the "This connection uses the following items" list, double-click **Internet Protocol (TCP/IP)**. The Internet Protocol (TCP/IP) Properties dialog box is displayed.

Figure 130: Internet Protocol (TCP/IP) Properties dialog box

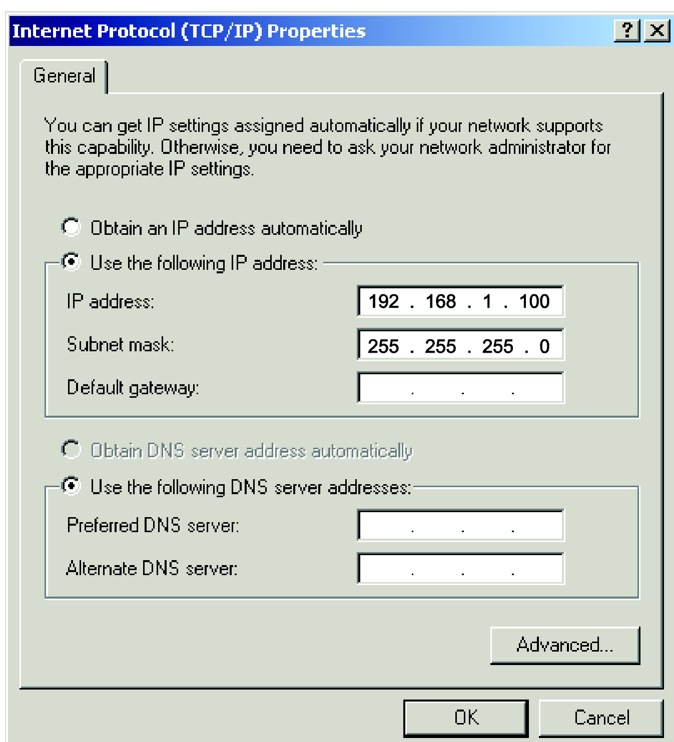


5. Select **Use the following IP address**. The option for "Use the following DNS server addresses" is automatically selected.
6. Set the IP address. If the IP address and subnet mask fields:
   - **Contain values:** Record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the instrument's LAN settings (on page B-8).
   - **Are blank:** In the IP address field, enter 192.168.1.100. In the subnet mask field, enter 255.255.255.0. These will be used to configure the LAN settings of the instrument.
1. Click **OK**.
2. Click **OK** to close the Local Area Connection Properties dialog box.
3. Close the Network Connections window.

Windows Vista: To disable DHCP:

1. Click **Start > Control Panel**.
2. Click **Network and Internet**.
3. Open **Network & Sharing Center**.
4. In the list, click **View Status** for the applicable connection. The Local Area Connection Status properties dialog box is displayed.
5. Click **Properties**. Windows displays a permissions message.
6. If you are logged in as administrator, click **Continue.** If you are not logged in as administrator, enter the administrator's password to continue. The network connection properties dialog box is displayed.
7. Double-click **Internet Protocol Version 4 (TCP/IPv4)** in the items list. The Internet Protocol Version 4 (TCP/IPv4) Properties dialog box is displayed.

Figure 131: Internet Protocol (TCP/IP) Properties dialog box

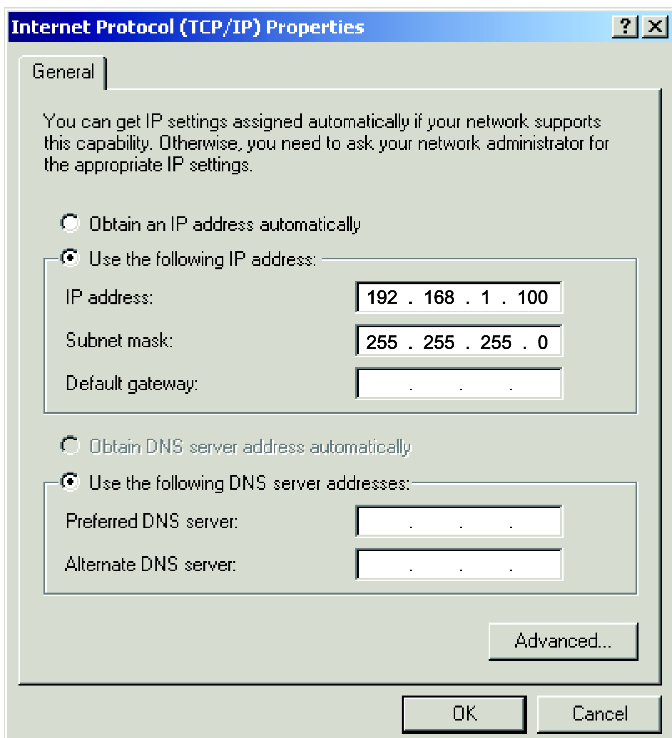


8. Select **Use the following IP address**. The option for "Use the following DNS server addresses" is automatically selected.
9. Set the IP address. If the IP address and subnet mask fields:
    - **Contain values:** Record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the instrument's LAN settings (on page B-8).
    - **Are blank:** In the IP address field, enter 192.168.1.100. In the subnet mask field, enter 255.255.255.0. These will be used to configure the LAN settings of the instrument.
1. Click **OK** to close the Internet Protocol Version 4 (TCP/IPv4) Properties dialog box.
2. Click **OK** to close the Local Area Connection Properties dialog box.
3. Close the Network Connections window.

Windows 7: To disable DHCP:

1. Click **Start > Control Panel**.
2. Open **Network and Sharing Center**.
3. Click the Local Area Connection. The Local Area Connection Status dialog box is displayed.
4. In the items list, double-click **Internet Protocol Version 4 (TCP/IPv4)**. The Internet Protocol Version 4 (TCP/IPv4) Properties dialog box is displayed.

Figure 132: Internet Protocol (TCP/IP) Properties dialog box



5. Select **Use the following IP address**. The option for "Use the following DNS server addresses" is automatically selected.
6. Set the IP address. If the IP address and subnet mask fields:
   - **Contain values:** Record the IP address, subnet mask, default gateway, and DNS servers to use in Step 3: Configure the instrument's LAN settings (on page B-8).
   - **Are blank:** In the IP address field, enter 192.168.1.100. In the subnet mask field, enter 255.255.255.0. These will be used to configure the LAN settings of the instrument.
1. Click **OK** to close the Internet Protocol Version 4 (TCP/IPv4) Properties dialog box.
2. Click **OK** to close the Local Area Connection Properties dialog box.
3. Close the Network Connections window.

Step 3: Configure the instrument's LAN settings

NOTE

These steps assume that you are making all the settings in the order shown here. If you only change one or a few settings, be aware that you need to apply the settings before they will be in effect. To apply the settings, from the **LAN CONFIG** menu, select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To configure the Series 3700A using the front panel:

1. Press the **MENU** key to display the MAIN MENU.
2. Use the navigation wheel ◷ to select **LAN**. The LAN CONFIG menu is displayed.
3. Change the IP address assignment method:
   a. Select **CONFIG > METHOD > MANUAL**, and then press the **ENTER** key.
   b. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
4. Enter the IP address using the LAN CONFIG menu:
   a. Select **CONFIG > IP-ADDRESS**.
   b. Refer to the recorded computer's IP address (Step 1: Identify and record the existing IP configuration (on page B-2)). A portion of the computer's IP address is used as a base for the instrument's unique ID. Only the last three numbers (after the last decimal point) of the IP address will differ between the computer and the instrument. If the subnet mask is 255.255.255.0, the last three digits can be any value from 1 to 255.

      For example, the Internet Protocol (TCP/IP) Properties dialog box shows that the computer's IP address is 192.168.1.100 (see the figure titled "Internet protocol (TCP/IP) Properties dialog box" in Step 2: Disable DHCP to use the computer's existing IP address (on page B-4)). A unique IP address for the instrument might be 192.168.001.101.

NOTE

The instrument's IP address can have leading zeros, but the computer's IP address cannot.

   c. Use the navigation wheel ◷ to select and enter an appropriate IP address for the instrument. Be sure to record the instrument's IP address to use in Step 5: Access the instrument's web page (on page B-9).
   d. Press **ENTER** key or navigation wheel ◷ to confirm the changes.
   e. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.

5. Change the subnet mask from the LAN CONFIG menu:
   a. Select **CONFIG > SUBNETMASK**, and then press the **ENTER** key. The SUBNETMASK menu item is to the right of GATEWAY. Use the navigation wheel to scroll through the options.
   b. Modify the SUBNETMASK value to match the computer settings recorded earlier (or `255.255.255.000` if DHCP Enabled = YES).
   c. Press the **ENTER** key or the navigation wheel when you are finished changing all the characters.
   d. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
6. From the **LAN MENU**, select **APPLY > YES**, and then press the **ENTER** key.

Step 4: Install the crossover cable

Connect the supplied crossover cable between the computer's NIC card and the LAN connector on the instrument's rear panel. There are multiple connectors on the Series 3700A rear panel. Be sure to connect to the LAN connection port.

NOTE

Connect the crossover cable into the same computer LAN port used during instrument configuration to ensure that the system is using the correct network card.

Step 5: Access the instrument's web page

1. Open a web browser on the host computer.
2. Enter the IP address of the instrument in the web browser address box. 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.

NOTE

If the web page does not open in the browser, see LAN troubleshooting suggestions (see "Verify connections and settings" on page 9-6).

Connecting to the LAN

Each device on the LAN (corporate or private) requires a unique IP address. Contact your corporate information technology (IT) department for details about obtaining an IP address before you deploy the Series 3700A on a corporate or private network.

CAUTION

Contact your corporate IT department for permission before you connect the Series 3700A to a corporate network.

Setting the LAN configuration method

There are two methods used to configure the LAN.

AUTO: Use the AUTO setting to allow the DHCP server to automatically set the LAN settings.

You do not need to set the LAN options manually. The DHCP server automatically configures the IP address, subnet mask, and the default gateway. To use this option, a DHCP server must be available on the LAN.

MANUAL: Use the MANUAL setting to manually configure the communication parameters.

The MANUAL setting requires you to configure the following:

- IP address
- Gateway
- Subnet mask

To select a LAN configuration method:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > METHOD**.
2. Select either **AUTO** or **MANUAL**.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Setting the IP address

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.

To set the IP address (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > IP-ADDRESS**.
2. Turn the navigation wheel to select and enter a valid IP address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Setting the gateway

NOTE

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

To set the gateway (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > GATEWAY**.
2. Turn the navigation wheel ◷ to select and enter a valid gateway address for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Setting the subnet mask

NOTE

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

To set the subnet mask (when LAN configuration method is set to MANUAL):

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > SUBNETMASK**.
2. Turn the navigation wheel ◷ to select and enter a valid subnet mask for the instrument.
3. Press the **ENTER** key to confirm the changes.
4. Press the **EXIT (LOCAL)** key to return to the LAN CONFIG menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Configuring the domain name system (DNS)

The domain name system (DNS) lets you type a domain name in the address bar to connect to the instrument. If you use DNS, you can use a name instead of an IP address.

Example:

```
Model3700AS.XYZcompany.com
```

NOTE

Contact your corporate information technology (IT) department to learn more about DNS. If a DNS server is not part of the LAN infrastructure, this setting is not used.

To enable or disable DNS host name verification:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > DNS > VERIFY**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. When enabled, the instrument performs a DNS lookup to verify the DNS host name matches the value specified in the lan.config.dns.hostname (on page 8-269) attribute.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To enable or disable DNS registration:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > DNS > DYNAMIC**.
2. Turn the navigation wheel to select either **ENABLE** or **DISABLE**. DNS registration works with the DHCP to register the host name specified in the `lan.config.dns.hostname` attribute with the DNS server.
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
5. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

To set the DNS server IP addresses:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > DNS**.
2. Turn the navigation wheel to select either **DNS-ADDRESS1** or **DNS-ADDRESS2**.
3. Press the **ENTER** key.
4. Turn the navigation wheel to select and enter a valid IP address for the DNS server.
5. Press the **ENTER** key.
6. Press the **EXIT (LOCAL)** key twice to return to the LAN CONFIG menu.
7. Select **APPLY_SETTINGS > YES**, and then press the **ENTER** key.

LAN speeds

Another characteristic of the LAN is speed. The Series 3700A negotiates with the host computer and other LXI-compliant devices on the LAN to transmit data at the highest speed possible. LAN speeds must be configured to match the speed of the other instruments on the network.

To set the LAN speed:

1. From the front panel, press the **MENU** key and select **LAN > CONFIG > SPEED**.
2. Turn the navigation wheel to select either **10 Mbps** or **100 Mbps.**
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the previous menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Duplex mode

The duplex mode is based on the LAN configuration. There are two settings:

- **Half-duplex:** Allows communications in both directions, but only one direction is active at a time (not simultaneously).
- **Full:** Permits communications in both directions simultaneously.

To set the duplex mode:

1. From the front panel, press **MENU** key and select **LAN > CONFIG > DUPLEX**.
2. Turn the navigation wheel to select either **HALF** or **FULL.**
3. Press the **ENTER** key.
4. Press the **EXIT (LOCAL)** key once to return to the LAN CONFIG menu.
5. Select **APPLY > YES**, and then press the **ENTER** key.

Viewing LAN status messages

To view the LAN status messages:

1. From the front panel, press the **MENU** key and select **LAN > STATUS > CONFIG/FAULT**.
2. Press the **ENTER** key.

Figure 133: LAN CONFIG/FAULT

There are two types of LAN status messages:

- **LAN fault messages:** Communicate issues related to physical connectivity.
- **LAN configuration messages:** Communicate issues or events related to configuration.

The following table displays possible fault and configuration messages.

LAN CONFIG/FAULT messages

LAN message type	Possible messages
LAN fault	Could not acquire IP address
	Duplicate IP address detected
	DHCP lease lost
	Lan Cable Disconnected
LAN configuration	Starting DHCP Configuration
	DHCP Server Not Found
	DHCP configuration started on xxx.xxx.xxx.xxx
	Searching for DNS server(s)
	Starting DLLA Configuration
	DLLA Failed
	DLLA configuration started on xxx.xxx.xxx.xxx
	Starting Manual Configuration
	Manual configuration started on xxx.xxx.xxx.xxx
	Closed

Viewing the network settings

To view the active network settings:

1. From the front panel, press the **MENU** key, and then select **LAN > STATUS**.
2. Use the navigation wheel to select one of the following network settings:
   - **IP-ADDRESS**
   - **GATEWAY**
   - **SUBNET-MASK**
   - **METHOD**
   - **DNS**
   - **MAC-ADDRESS**

1. Press the **ENTER** key to view the active setting.
2. Press the **EXIT (LOCAL)** key once to return to the STATUS menu.

Confirming the active speed and duplex negotiation

The Series 3700A automatically detects the speed and duplex negotiation active on the LAN. Once the speed and duplex negotiation is detected, the instrument automatically adjusts its own settings to match the LAN settings.

To confirm the active LAN speed and duplex mode:

1. From the front panel, press the **MENU** key.
2. Select **LAN > STATUS**.
3. Use the navigation wheel to select one of the following:
   - **SPEED**
   - **DUPLEX**

1. Press the **ENTER** key to view the active setting.
2. Press the **EXIT (LOCAL)** key once to return to the STATUS menu

Confirming port numbers

To view the port number assigned to each remote interface protocol:

1. From the front panel, press the **MENU** key, and then select **LAN > STATUS > PORT**.
2. Use the navigation wheel to select one of the following:
   - **RAW-SOCKET**
   - **TELNET**
   - **VXI-11**
   - **DST**

1. Press the **ENTER** key to view the port number.
2. Press the **EXIT (LOCAL)** key once to return to the PORT menu.

The following table displays the remote interface protocols supported by the Series 3700A and their assigned port numbers.

Port number

Command interface	Port number
Raw socket	5025
Telnet	23
VXI-11	1024
DST (dead socket termination)	5030

Selecting a LAN interface protocol

This section provides details about how to select a remote interface protocol to connect to the Series 3700A. The Series 3700A provides three LAN interfaces with three associated LAN protocols (each interface uses a different protocol). Select the interface based on the protocol needed. The dead socket termination interface (DST) is provided to solve connection problems; it is not a protocol choice.

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.

Raw socket connection

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

Dead socket connection

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

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

Telnet connection

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

The Series 3700A 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.

Configuring a Telnet connection

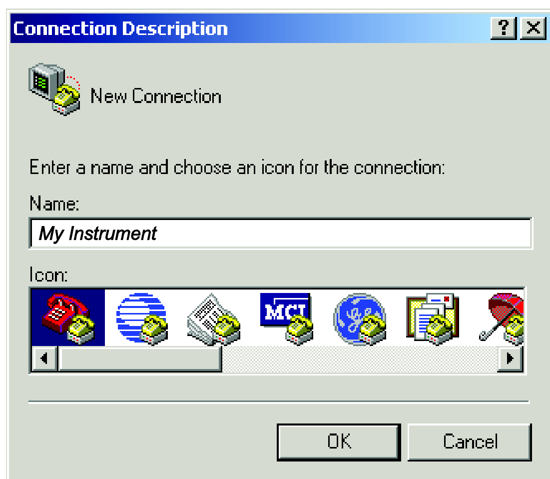
NOTE

This procedure uses HyperTerminal™, which is available with the Microsoft® Windows® XP operating system. Consult the help system for your version of Microsoft Windows to identify a compatible tool.

To connect with the Series 3700A using HyperTerminal on a Windows XP system:

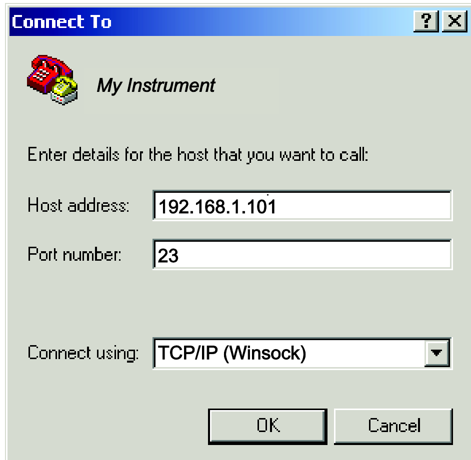
1. On the host computer, click **Start > Accessories > Communications > HyperTerminal**. The Connection Description dialog box opens.

Figure 134: Connection description dialog box



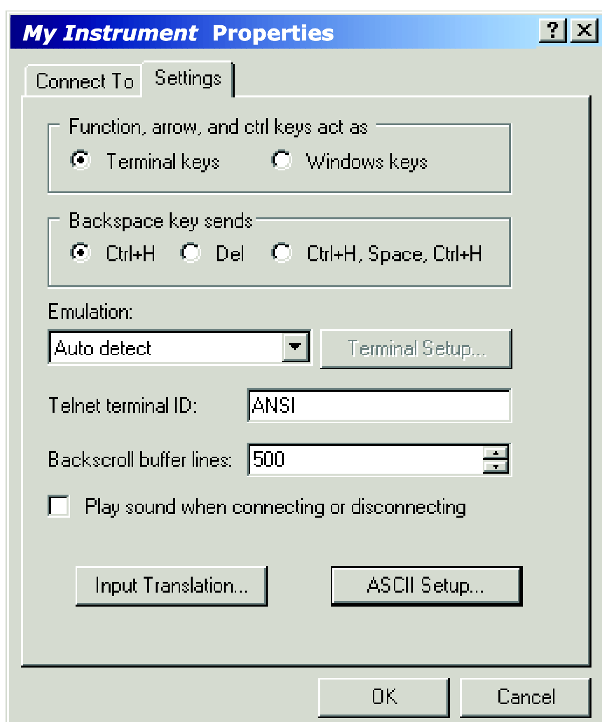
2. Type a name to identify the connection (for example, `My Instrument`), and then click **OK**.
3. In the Connect To dialog box, click the **Connect using** list. Select **TCP/IP (Winsock)**.

Figure 135: Connect To dialog box



4. In the Host address field, type the instrument's IP address (for example, `192.168.1.101`) .
5. Type 23 in the **Port number** field, and then click **OK**. The HyperTerminal program window is displayed.
6. From the HyperTerminal program window, click **File > Properties**.
7. In the Properties dialog box, click the **Settings** tab.

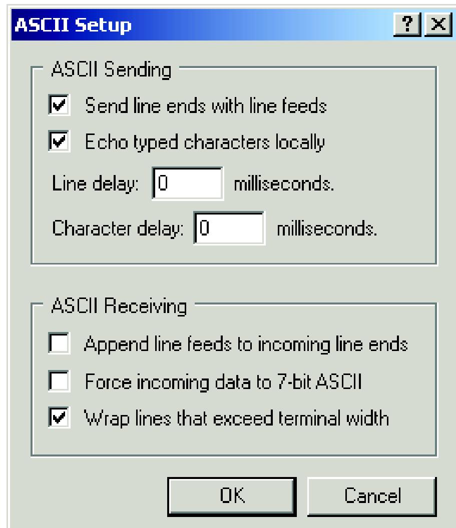
Figure 136: Properties dialog box



8. Click **ASCII Setup**. The ASCII Setup dialog box is displayed.

9. From the ASCII Setup dialog box, select the following options:
   - **Send line ends with line feeds**
   - **Echo typed characters locally**

Figure 137: ASCII Setup dialog box



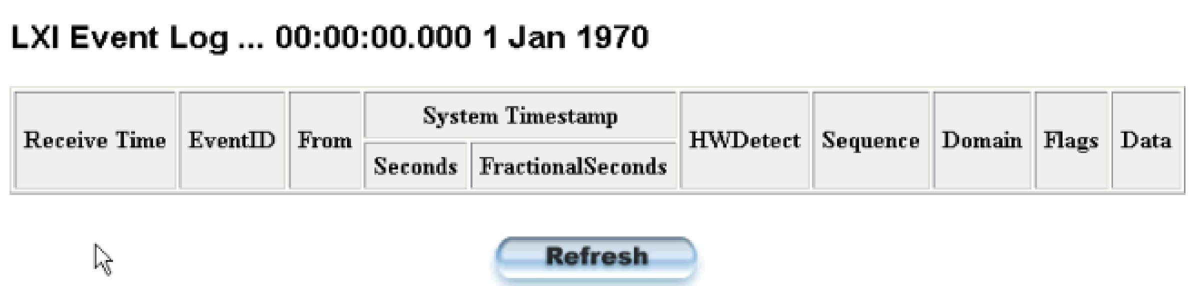
1. Click **OK** in the ASCII Setup dialog box. The Properties dialog box is displayed.
2. Click **OK** in the Properties dialog box.

Use the HyperTerminal window to interact directly with the instrument.

Logging LAN trigger events in the event log

You can use the event log to record all LXI triggers generated and received by the Series 3700A, and you can view the event log using any command interface or the embedded web interface. The following figure shows the view of the LXI event log from the embedded web interface.

Figure 138: 3700A LXI event log



The timestamp, event identifier, IP address, and the domain name identify the incoming and outgoing LXI trigger packets. The following table provides detailed descriptions for the columns in the event log.

Event log descriptions

Column title	Description	Example
Received Time	Displays the date and time that the LAN trigger occurred in UTC, 24-hour time	06:56:28.000 8 May 2011
Event ID	Identifies the `lan.trigger[N]` that generates an event	`LAN0 = lan.trigger[1]` `LAN1 = lan.trigger[2]` `LAN2 = lan.trigger[3]` `LAN3 = lan.trigger[4]` `LAN4 = lan.trigger[5]` `LAN5 = lan.trigger[6]` `LAN6 = lan.trigger[7]` `LAN7 = lan.trigger[8]`
From	Displays the IP address for the device that generates the LAN trigger	localhost 192.168.5.20
Timestamp	A timestamp that identifies the time the event occurred; the timestamp uses the following: • PTP timestamp • Seconds • Fractional seconds The Series 3700A does not support the IEEE Std 1588 standard; the values in this field are always 0 (zero)	
HWDetect	Identifies a valid LXI trigger packet	LXI
Sequence	Each instrument maintains independent sequence counters: • One for each combination of UDP multicast network interface and UDP multicast destination port • One for each TCP connection	
Domain	Displays the LXI domain number; the default value is 0 (zero)	0 1523
Flags	Contain data about the LXI trigger packet; values are: • 1 - Error • 2 - Retransmission • 4 - Hardware • 8 - Acknowledgments • 16 - Stateless bit	
Data	The values for this are always 0 (zero)	

Accessing the event log from the command interface

You can access the event log from any remote command interface. The event log must be enabled before LXI trigger events can be viewed. To enable the event log, send:

```
eventlog.enable = 1
```

To view the event log from a remote interface, send:

```
print(eventlog.all())
```

This command outputs one or more strings similar to the following:

```
14:14:02.000 17 Jun 2008, LAN0, 10.80.64.191, LXI, 0, 1213712000, not
available, 0, 0x10,0x00
```

The string displays the same information as the web interface. Commas separate the fields. The fields output in the following order:

- Received time (UTC time)
- Event ID
- From (Sender)
- HWDetect / version
- Domain
- Sequence number
- Timestamp (PTP time)
- Epoch (from 1588)
- Flags
- Data

See the table in Logging LAN trigger events in the event log (on page B-18) for detailed descriptions.

Appendix C

Calibration

In this appendix:

Verification

Use the procedures in this section to verify that the Keithley Instruments Model 3706A System Switch/Multimeter's accuracy is within the limits stated in the instrument's one-year accuracy specifications. Verifying the accuracy of your Model 3706A is recommended:

- When you first receive the instrument to make sure that it was not damaged during shipment
- To verify that the unit meets factory specifications
- To determine if calibration is required
- Following calibration to make sure that calibration was performed properly

WARNING

The information in this section is intended for qualified service personnel only. Do not attempt these procedures unless you are qualified to do so.

Some of these procedures may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

For the plug-in modules, the maximum common-mode voltage (voltage between any plug-in module terminal and chassis ground) is 300V DC or 300V RMS. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

NOTE

If the instrument is still under warranty and its performance is outside specified limits, contact your Keithley Instruments representative or the factory to determine the correct course of action.

Verification test requirements

Be sure that you perform these verification tests:

- Under the proper environmental conditions
- After the specified warmup period
- Using the correct line voltage
- Using the proper test equipment
- Using the specified output signal and reading limits

Environmental conditions

Conduct the verification procedures in a location that has:

- An ambient temperature of 18 °C to 28 °C (65 °F to 82 °F)
- A relative humidity of less than 80%, unless otherwise noted

Warmup period

NOTE

At the factory, instruments are calibrated without any switch cards installed and all slots are covered with blank slot covers. The slot covers come installed on the instrument when it is shipped.

If it is more convenient to calibrate the instrument with switch cards installed, make sure all channels are open and any empty slots are covered with blank slot covers.

Allow the System Switch/Multimeter to warm up for at least two hours before performing calibration.

If the instrument has been subjected to temperature extremes (those outside the ranges stated in Environmental conditions (on page C-2)), allow extra time for the instrument's internal temperature to stabilize. Typically, you need to allow one extra hour to stabilize an instrument that is 10 °C (18 °F) outside the specified temperature range.

Also, allow the test equipment to warm up for the minimum time specified by the manufacturer.

Line power

The Model 3706A requires a line voltage of 100 V to 240 V (±10%), and a line frequency of 50 Hz or 60 Hz.

NOTE

The instrument automatically senses the line frequency at power-up.

Recommended test equipment

The following table summarizes recommended verification equipment. You can use alternate equipment if that equipment has specifications equal to or greater than those listed in the table. Note, however, that test equipment uncertainty will add to the uncertainty of each measurement. Generally, test equipment uncertainty should be at least four times better (more accurate) than corresponding Model 3706A specifications.

NOTE

The Keithley Instruments Model 3706-190 backplane connector board is an accessory that can be used to make connections to the calibrator. Additional boards, such as a 4-wire short or the discrete resistors, would also be convenient to eliminate rewiring for different setups used in verification.

Manufacturer	Model	Description	Used for:	Uncertainty
Fluke	5700	Calibrator	All DCV, ACV, DCI, ACI, and resistance	See NOTE.
Fluke	5725	Amplifier	High voltage, high current	See NOTE.
HP	3458	DMM	10µA, 100µA DCI range	See NOTE.
Agilent	33220A	Function generator	Frequency	See NOTE.
N/A	N/A	4-wire short	DCV, resistance zeros	N/A
N/A	N/A	1 Ohm discrete resistor	1 Ohm range	+/- 20ppm
N/A	N/A	10 Ohm discrete resistor	10 Ohm range	+/- 20ppm

NOTE

Refer to the manufacturer's specifications to calculate the uncertainty, which will vary for each test point.

Verification limits

The verification limits stated in this section have been calculated using only the Model 3706A one-year accuracy specifications, and they do not include test equipment uncertainty. If a particular measurement falls outside the allowable range, recalculate new limits based both on the Model 3706A specifications and corresponding test equipment specifications.

Example reading limit calculation

The following is an example of how reading limits have been calculated. Assume you are testing the 10V DC range using a 10V input value. Using the Model 3706A one-year accuracy specification for 10V DC of ± (25ppm of reading + 2ppm of range), the calculated limits are:

Reading limits = 10V ± [(10V × 25ppm) + (10V × 2ppm)]

Reading limits = 10V ± (0.00025 + 0.00002)

Reading limits = 10V ± 0.00027V

Reading limits = 9.99973V to 10.00027V

Calculating resistance reading limits

Resistance reading limits must be recalculated based on the actual calibration resistance values supplied by the equipment manufacturer. Calculations are performed in the same manner as shown in the preceding example, using the actual calibration resistance values instead of the nominal values in the example when performing your calculations.

For example, assume that you are testing the 10 kΩ range using an actual 10.03 kΩ calibration resistance value. Using Model 3706A one-year 10 kΩ range accuracy of ± (60 ppm of reading + 4 ppm of range), the calculated reading limits are:

- Reading limits = 10.03 kΩ ± [(10.03 kΩ x 60 ppm) + (10.03 kΩ x 6 ppm)]
- Reading limits = 10.03 kΩ ± [(0.000618) + (0.0000618)]
- Reading limits = 10.03 kΩ ± 0.0006798
- Reading limits = 10.0293202 kΩ to 10.0306798 kΩ

Restoring factory defaults

To restore the instrument to its factory front panel (bench) defaults before performing the verification procedures:

1. Press the **MENU** key.
2. Turn the navigation wheel to highlight **SETUP** and then press the **ENTER** key.
3. Turn the navigation wheel to highlight **RESET** and then press the **ENTER** key.

Performing the verification test procedures

The following topics provide a summary of verification test procedures, as well as items to take into consideration before performing any verification test.

Test summary

- Verifying DC voltage (on page C-5)
- Verifying AC voltage (on page C-7)
- Verifying DC current 10 µA to 100 µA ranges (on page C-9)
- Verifying DC current 1 mA to 3 A ranges (on page C-10)
- Verifying AC current 1 mA to 3 A ranges (on page C-12)
- Verifying frequency (on page C-14)
- Verifying 4-wire resistance (on page C-15)
- Verifying 2-wire resistance (on page C-16)
- Verifying dry circuit resistance (on page C-17)
- Verifying 1-ohm and 10-ohm resistance ranges (on page C-19)
- Verifying zeros using a 4-wire short (on page C-20)

If the Model 3706A is not within specifications and not under warranty, calibrate the unit.

Test considerations

When performing the verification procedures:

- Be sure to restore factory front panel defaults as outlined in Restoring factory defaults (on page C-4).
- Make sure that the test equipment is properly warmed up and connected to the Model 3706A terminals.
- Be sure the test equipment is set up for the proper function and range.
- Do not connect test equipment to the Model 3706A through a scanner, multiplexer, or other switching equipment.

WARNING

The input/output terminals of the digital multimeter (DMM) and switch cards are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the DMM or switch card terminals to CAT II, CAT III, or CAT IV circuits.

Connections of the DMM or switch card terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

Model 3706A verification tests

Perform these tests to verify the accuracy of your Model 3706A at the analog backplane connector.

Verifying DC voltage

Check DC voltage accuracy by applying accurate voltages from the DC voltage calibrator to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION

Do not exceed 300 V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify DC voltage accuracy:

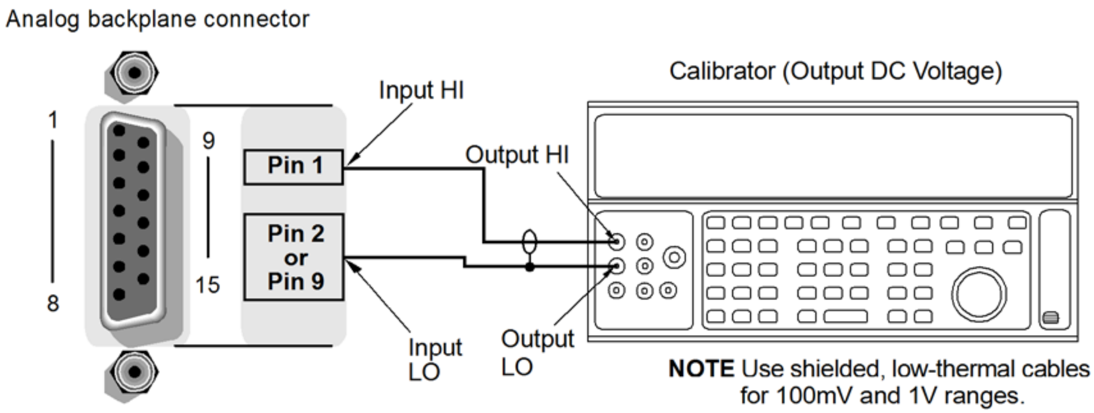
NOTE

Use shielded, low-thermal connections when testing the 100 mV and 1 V ranges to avoid errors caused by noise or thermal effects. Connect the shield to the calibrator’s output LO terminal.

1. Connect the Model 3706A HI and LO INPUT pins to the DC voltage calibrator as shown in the "DC voltage verification" below.
2. Select the DC volts function.
3. Set the Model 3706A to the 100 mV range.

4. If relative offsetis needed, set the calibrator output to 0.00000 mVDC and allow the reading to settle.
5. Enable the Model 3706A relative offset mode.
6. Source positive and negative full-scale and half-scale voltages for each of the ranges listed in the table below. For each voltage setting, be sure that the reading is within stated limits.

Figure 139: DC voltage verification



DC voltage verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on the published specifications (see Example reading limit calculation (on page C-3)).

Connect to the Fluke 5700A Calibrator				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Rel Series 3700	1.00E-01	0.00E+00	N/A	N/A
Verify DCV 100mV	1.00E-01	1.00E-01	9.999610E-02	1.000039E-01
Verify DCV 100mV	1.00E-01	5.00E-02	4.999760E-02	5.000240E-02
Verify DCV 100mV	1.00E-01	-5.00E-02	-5.000240E-02	-4.999760E-02
Verify DCV 100mV	1.00E-01	-1.00E-01	-1.000039E-01	-9.999610E-02
Rel Series 3700	1.00E+00	0.00E+00	N/A	N/A
Verify DCV 1V	1.00E+00	1.00E+00	9.999680E-01	1.000032E+00
Verify DCV 1V	1.00E+00	5.00E-01	4.999830E-01	5.000170E-01
Verify DCV 1V	1.00E+00	-5.00E-01	-5.000170E-01	-4.999830E-01
Verify DCV 1V	1.00E+00	-1.00E+00	-1.000032E+00	-9.999680E-01
Verify DCV 10V	1.00E+01	1.00E+01	9.999730E+00	1.000027E+01
Verify DCV 10V	1.00E+01	5.00E+00	4.999855E+00	5.000145E+00
Verify DCV 10V	1.00E+01	0.00E+00	-2.000000E-05	2.000000E-05
Verify DCV 10V	1.00E+01	-5.00E+00	-5.000145E+00	-4.999855E+00
Verify DCV 10V	1.00E+01	-1.00E+01	-1.000027E+01	-9.999730E+00

Connect to the Fluke 5700A Calibrator				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Verify DCV 100V	1.00E+02	1.00E+02	9.999540E+01	1.000046E+02
Verify DCV 100V	1.00E+02	5.00E+01	4.999740E+01	5.000260E+01
Verify DCV 100V	1.00E+02	0.00E+00	-6.000000E-04	6.000000E-04
Verify DCV 100V	1.00E+02	-5.00E+01	-5.000260E+01	-4.999740E+01
Verify DCV 100V	1.00E+02	-1.00E+02	-1.000046E+02	-9.999540E+01
Verify DCV 300V	3.00E+02	3.00E+02	2.999862E+02	3.000138E+02
Verify DCV 300V	3.00E+02	1.50E+02	1.499922E+02	1.500078E+02
Verify DCV 300V	3.00E+02	0.00E+00	-1.800000E-03	1.800000E-03
Verify DCV 300V	3.00E+02	-1.50E+02	-1.500078E+02	-1.499922E+02
Verify DCV 300V	3.00E+02	-3.00E+02	-3.000138E+02	-2.999862E+02

Verifying AC voltage

Check AC voltage accuracy by applying accurate voltages from the AC voltage calibrator to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

⚠ CAUTION

Do not exceed 300 V peak between INPUT HI and INPUT LO, or 8×10^7 VHz input, because instrument damage may occur.

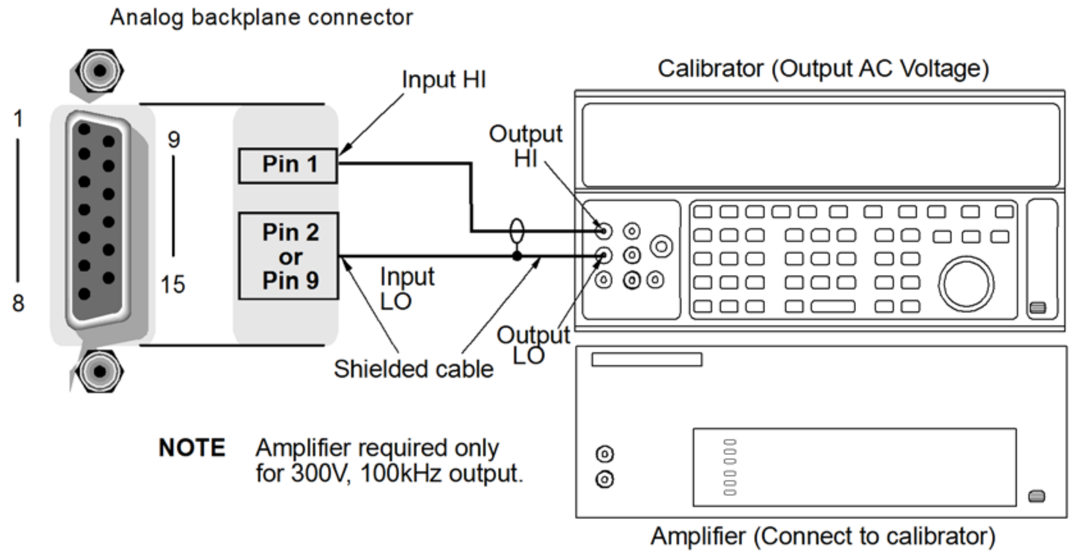
To verify AC voltage accuracy:

NOTE

Use shielded, low-thermal connections when testing the 100 mV and 1 V ranges to avoid errors caused by noise or thermal effects. Connect the shield to the calibrator's output LO terminal.

1. Connect the Model 3706A HI and LO INPUT pins to the DC voltage calibrator as shown in "AC voltage verification" below.
2. Select the AC volts function.
3. Set the Model 3706A to the 100 mV range. Make sure that relative offset is disabled.
4. Source AC voltages for each of the frequencies and ranges are summarized in the ACV verification data (on page C-8) table. For each setting, be sure that the reading is within stated limits.
5. Repeat steps 3 and 4 for each item in the table.

Figure 140: AC voltage verification



ACV verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

Connect to the Fluke 5700A calibrator				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Verify ACV 100mV @ 20Hz	1.00E-01	1.00E-01	9.897000E-02	1.010300E-01
Verify ACV 100mV @ 1kHz	1.00E-01	1.00E-01	9.992000E-02	1.000800E-01
Verify ACV 100mV @ 50kHz	1.00E-01	1.00E-01	9.984000E-02	1.001600E-01
Verify ACV 100mV @ 100kHz	1.00E-01	1.00E-01	9.932000E-02	1.006800E-01
Verify ACV 1V @ 20Hz	1.00E+00	1.00E+00	9.992000E-01	1.000800E+00
Verify ACV 1V @ 1kHz	1.00E+00	1.00E+00	9.992000E-01	1.000800E+00
Verify ACV 1V @ 50kHz	1.00E+00	1.00E+00	9.984000E-01	1.001600E+00
Verify ACV 1V @ 100kHz	1.00E+00	1.00E+00	9.932000E-01	1.006800E+00
Verify ACV 10V @ 1kHz	1.00E+01	1.00E+01	9.992000E+00	1.000800E+01
Verify ACV 10V @ 50kHz	1.00E+01	1.00E+01	9.984000E+00	1.001600E+01
Verify ACV 10V @ 100kHz	1.00E+01	1.00E+01	9.932000E+00	1.006800E+01
Verify ACV 100V @ 1kHz	1.00E+02	1.00E+02	9.992000E+01	1.000800E+02
Verify ACV 100V @ 50kHz	1.00E+02	1.00E+02	9.984000E+01	1.001600E+02
Verify ACV 100V @ 100kHz	1.00E+02	1.00E+02	9.932000E+01	1.006800E+02

Connect to the Fluke 5700A calibrator				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Verify ACV 300V @ 1kHz	3.00E+02	3.00E+02	2.997600E+02	3.002400E+02
Verify ACV 300V @ 50kHz	3.00E+02	3.00E+02	2.995200E+02	3.004800E+02

Connect to the Fluke 5725A amplifier				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Verify ACV 300V @ 100kHz	3.00E+02	3.00E+02	2.979600E+02	3.020400E+02

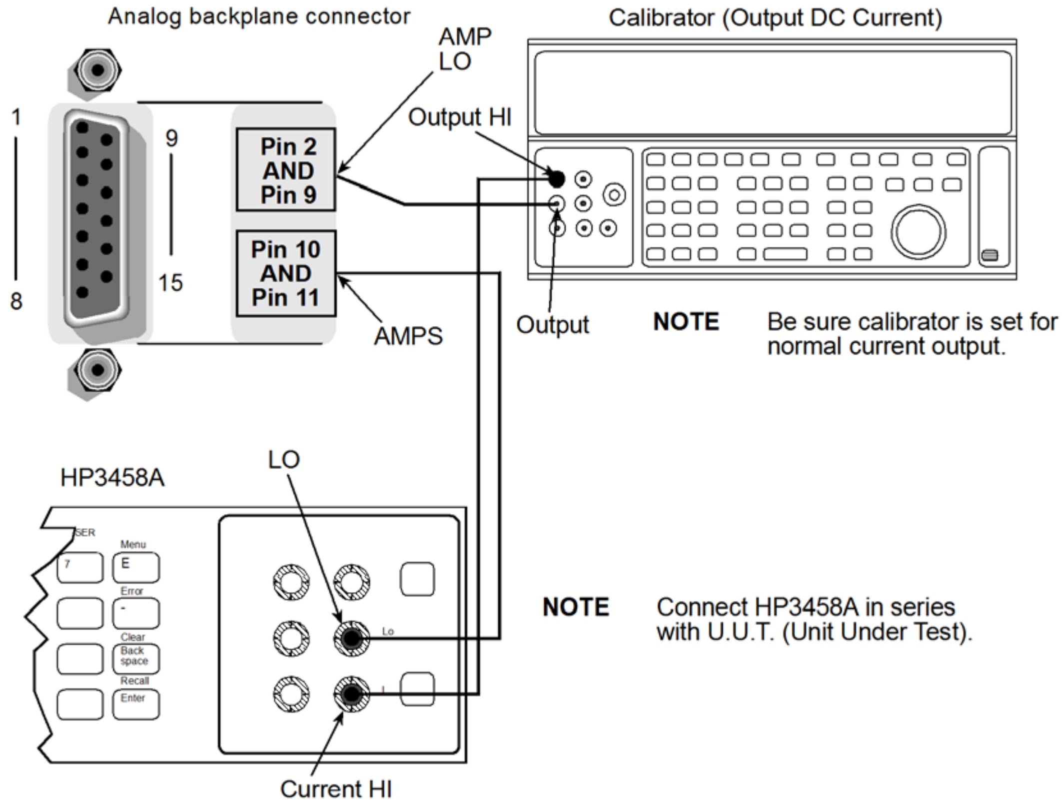
Verifying DC current 10 µA to 100 µA ranges

Check DC current accuracy by applying accurate current from the DC current calibrator to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

To verify DC current accuracy:

1. Set up the Model 3706A for DC current and the range being tested. Make sure relative offset is disabled.
2. Verify the zero test point for each range without any connection to the equipment and verify that the readings fall within specified limits.
3. Connect the Model 3706A AMPS and LO INPUT pins to the DC current calibrator as shown in the "DC current verification 10 uA to 100 uA ranges diagram" below.
4. Set up the HP3458A to the DC current function and range.
5. Set the calibrator to source zero current and rel both the Model 3706A and the HP3458A.
6. Source DC current for each of the test points summarized in the DC voltage verification data (on page C-6) table. For each setting, be sure that the reading is within stated limits.

Figure 141: DC current verification 10µA to 100µA ranges



DC current verification data 10 µA to 100 µA ranges

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

Connect HP3458A in series with 5700 calibrator				
Description	**Range (A)**	**Test point(A)**	**Lower limit (A)**	**Upper limit (A)**
Verify 10 µA Zero	1.00E-05	0.00E+00	-3.000000E-10	3.000000E-10
Verify DC Curr 10 µA	1.00E-05	1.00E-05	9.994700E-06	1.000530E-05
Verify DC Curr 10 µA	1.00E-05	-1.00E-05	-1.000530E-05	-9.994700E-06
Verify 100 µA Zero	1.00E-04	0.00E+00	-3.000000E-09	3.000000E-09
Verify DC Curr 100 µA	1.00E-04	1.00E-04	9.994910E-05	1.000509E-04
Verify DC Curr 100 µA	1.00E-04	-1.00E-04	-1.000509E-04	-9.994910E-05

Verifying DC current 1 mA to 3 A ranges

Check DC current accuracy by applying accurate current from the DC current calibrator to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

NOTE

The Fluke 5725A amplifier is only needed when verifying the 3 A range.

To verify DC current accuracy:

1. Connect the Model 3706A AMPS and LO INPUT pins to the DC current calibrator as shown in the "DC current verification 1 mA to 3 A ranges diagram" below, using the Keithley Instruments Model 3706-751 fixture cable.
2. Select the DC current function.
3. Set the Model 3706A to the applicable ranges. Make sure that relative offset is disabled.
4. Source DC current for each of the test points summarized in the DC current verification data table. For each setting, be sure that the reading is within stated limits.

Figure 142: DC current verification 1mA to 3A ranges

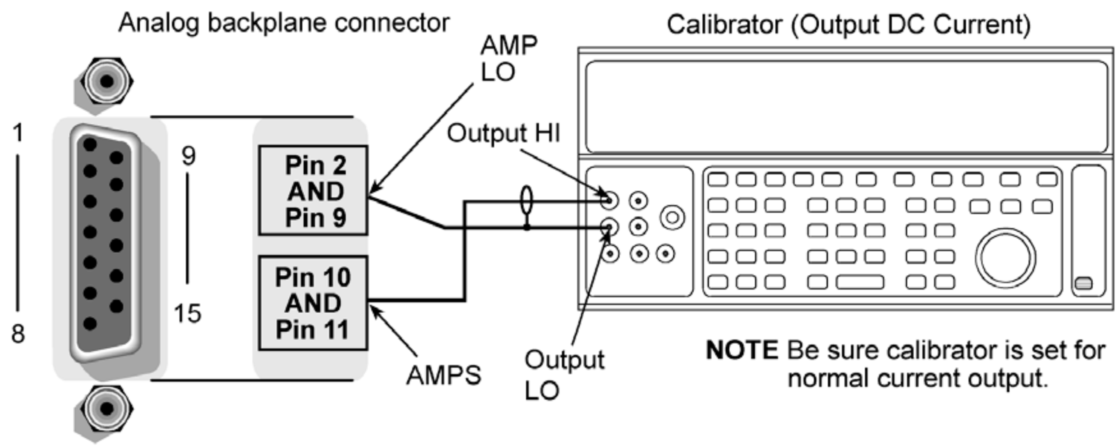
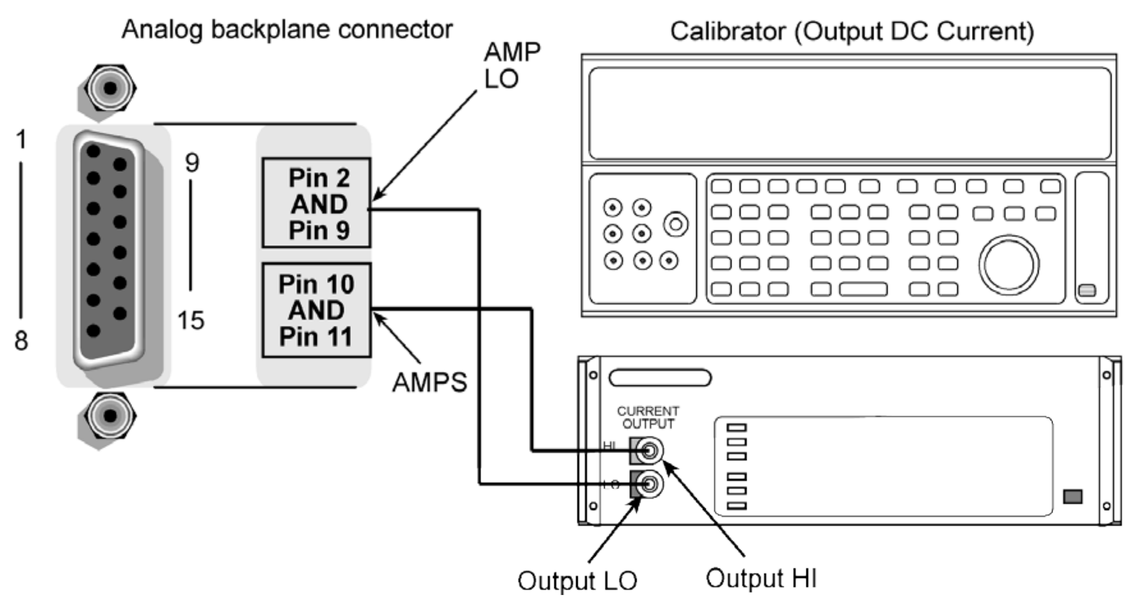


Figure 143: DC current verification 3A range diagram



DC current verification data 1 mA to 3 A ranges

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

Remove HP3458A, only connect the 5700				
Description	**Range (A)**	**Test point (A)**	**Lower limit (A)**	**Upper limit (A)**
Verify 1mA Zero	1.00E-03	0.00E+00	-9.000000E-09	9.000000E-09
Verify DC Curr 1mA	1.00E-03	1.00E-03	9.994910E-04	1.000509E-03
Verify DC Curr 1mA	1.00E-03	-1.00E-03	-1.000509E-03	-9.994910E-04
Verify 10mA Zero	1.00E-02	0.00E+00	-9.000000E-08	9.000000E-08
Verify DC Curr 10mA	1.00E-02	1.00E-02	9.994910E-03	1.000509E-02
Verify DC Curr 10mA	1.00E-02	-1.00E-02	-1.000509E-02	-9.994910E-03
Verify 100mA Zero	1.00E-01	0.00E+00	-9.000000E-07	9.000000E-07
Verify DC Curr 100mA	1.00E-01	1.00E-01	9.994910E-02	1.000509E-01
Verify DC Curr 100mA	1.00E-01	-1.00E-01	-1.000509E-01	-9.994910E-02
Verify DC Curr 1A	1.00E+00	1.00E+00	9.991900E-01	1.000810E+00
Verify DC Curr 1A	1.00E+00	-1.00E+00	-1.000810E+00	-9.991900E-01

Connect to the Fluke 5725A amplifier				
Description	**Range (A)**	**Test point (A)**	**Lower limit (A)**	**Upper limit (A)**
Verify DC Curr 3A	3.00E+00	3.00E+00	2.996355E+00	3.003645E+00
Verify DC Curr 3A	3.00E+00	-3.00E+00	-3.003645E+00	-2.996355E+00

Verifying AC current 1 mA to 3 A ranges

Check AC current accuracy by applying accurate current from the AC current calibrator at specific frequencies to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

To verify AC current accuracy:

1. Set up the Model 3706A for AC current and the range being tested. Make sure relative offset is disabled.
2. Source AC current for the 1 mA to 1 A range test points summarized in "AC current calibration diagram" below. For each setting, be sure that the reading is within stated limits.
3. Install the Fluke 5725A amplifier.
4. Source AC current for the 3 A range test points summarized in the AC current verification data 1mA to 1A ranges (see "AC current verification data 1 mA to 1 A ranges" on page C-14) table. Be sure that the 3 A readings are within stated limits.

Figure 144: AC current verification 1mA to 1A range

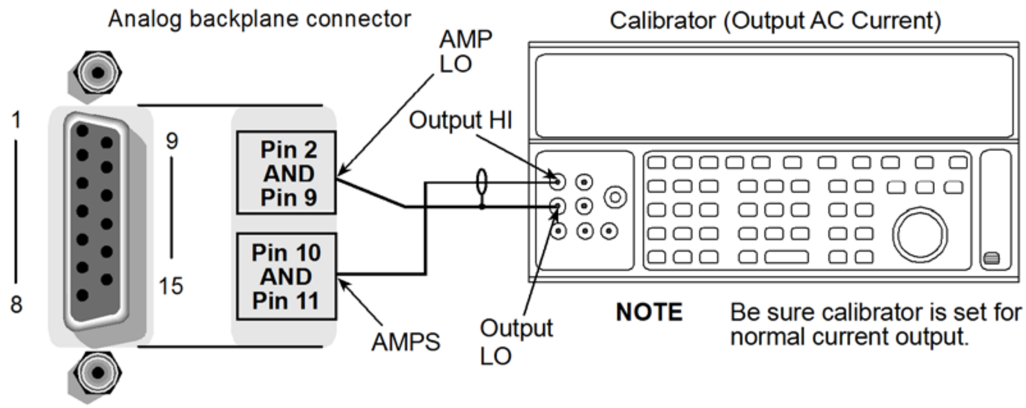
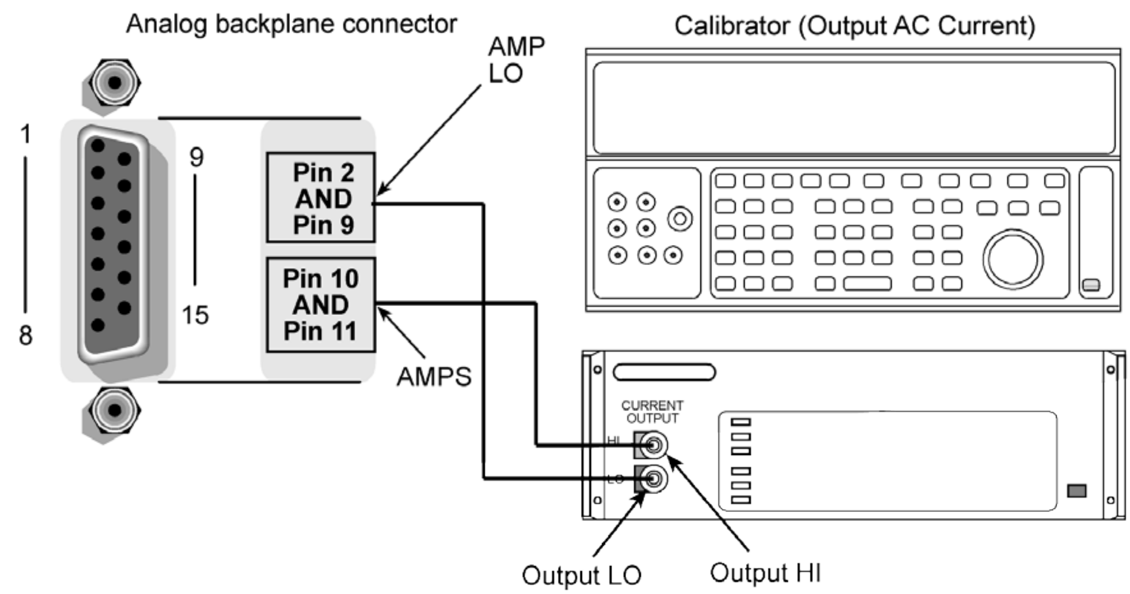


Figure 145: AC current verification 3A range



AC current verification data 1 mA to 1 A ranges

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

Connect to the Fluke 5700A calibrator				
Description	**Range (A)**	**Test point (A)**	**Lower limit (A)**	**Upper limit (A)**
Verify AC Curr 1mA @ 20Hz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03
Verify AC Curr 1mA @ 1kHz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03
Verify AC Curr 1mA @ 5kHz	1.00E-03	1.00E-03	9.989000E-04	1.001100E-03
Verify AC Curr 10mA @ 40Hz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 10mA @ 1kHz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 10mA @ 5kHz	1.00E-02	1.00E-02	9.989000E-03	1.001100E-02
Verify AC Curr 100mA @ 40Hz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 100mA @ 1kHz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 100mA @ 5kHz	1.00E-01	1.00E-01	9.989000E-02	1.001100E-01
Verify AC Curr 1A @ 40Hz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00
Verify AC Curr 1A @ 1kHz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00
Verify AC Curr 1A @ 5kHz	1.00E+00	1.00E+00	9.977000E-01	1.002300E+00

AC current verification data 3A range

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

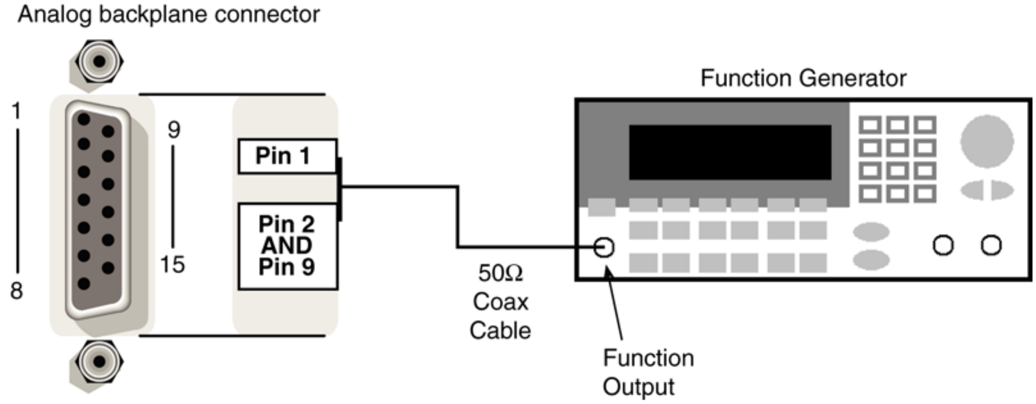
Connect to the Fluke 5725A amplifier				
Description	**Range (A)**	**Test point (A)**	**Lower limit (A)**	**Upper limit (A)**
Verify AC Curr 3A @ 40Hz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00
Verify AC Curr 3A @ 1kHz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00
Verify AC Curr 3A @ 5kHz	3.00E+00	3.00E+00	2.993100E+00	3.006900E+00

Verifying frequency

To verify the Model 3706A frequency function:

1. Connect the Agilent 33220A function generator to the Model 3706A INPUT pins.
2. Set the function generator to output a 1 kHz, 5 V RMS sine wave.
3. Select the Model 3706A frequency function by pressing the **FREQ** key.
4. Verify that each Model 3706A frequency reading is within the limits contained in the table contained in Frequency verification data (on page C-15).

Figure 146: Frequency verification



Frequency verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Example reading limit calculation (on page C-3)).

Connect the Agilent 33220A Generator				
Description	**Range (V)**	**Frequency (Hz)**	**Lower limit (Hz)**	**Upper limit (Hz)**
Verify Frequency 1kHz	1.00E+01	1.00E+03	9.999167E+02	1.000083E+03
Verify Frequency 10kHz	1.00E+01	1.00E+04	9.999167E+03	1.000083E+04
Verify Frequency 100kHz	1.00E+01	1.00E+05	9.999167E+04	1.000083E+05
Verify Frequency 250kHz	1.00E+01	2.50E+05	2.499797E+05	2.500203E+05
Verify Frequency 500kHz	1.00E+01	5.00E+05	4.999597E+05	5.000403E+05

Verifying 4-wire resistance

Check the normal resistance function by connecting accurate resistance values to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

CAUTION

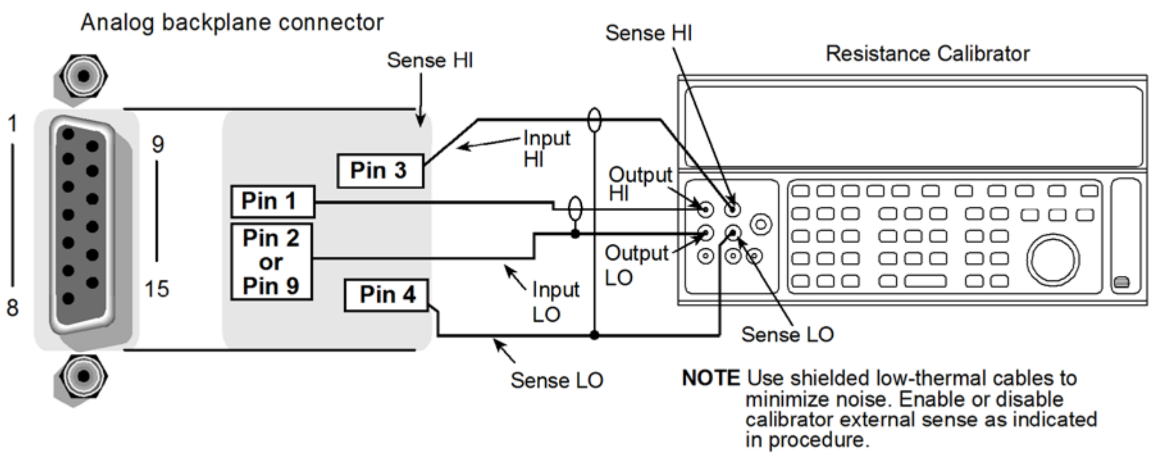
Do not exceed 300 V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify 4-wire resistance accuracy:

1. Using shielded, Teflon-insulated or equivalent cables in a 4-wire configuration, connect the Model 3706A INPUT and SENSE pins to the calibrator as shown for 100 Ω to 10 MΩ ranges.
2. Set the calibrator for 4-wire resistance with external sense on.
3. Select the Model 3706A 4-wire resistance function.
4. Select the SLOW integration rate with the **RATE** key.

5. Set the Model 3706A for the 100 Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 100 Ω and 1 kΩ range verification only. See Enabling/disabling offset-compensated ohms (on page 4-62) in the User's manual.
6. Recalculate reading limits based on actual calibrator resistance values.
7. Source the nominal full-scale resistance values for the 100 Ω to 10 MΩ ranges summarized in the 4-wire resistance verification data (on page C-16) table. Recalculate the limits based on the actual value of the resistor and verify the reading is within the calculated limits.

Figure 147: Resistance verification



4-wire resistance verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Calculating resistance reading limits (on page C-4)).

Connect to the Fluke 5700A calibrator				
Description	**Range (Ohms)**	**Test point (Ohms)**	**Lower limit (Ohms)**	**Upper limit (Ohms)**
Verify 4W Res 100 Ohm *	1.00E+02	1.00E+02	9.999310E+01	1.000069E+02
Verify 4W Res 1k Ohm	1.00E+03	1.00E+03	9.999360E+02	1.000064E+03
Verify 4W Res 10k Ohm	1.00E+04	1.00E+04	9.999360E+03	1.000064E+04
Verify 4W Res 100k Ohm	1.00E+05	1.00E+05	9.999360E+04	1.000064E+05
Verify 4W Res 1M Ohm	1.00E+06	1.00E+06	9.999560E+05	1.000044E+06
Verify 4W Res 10M Ohm	1.00E+07	1.00E+07	9.995900E+06	1.000410E+07

NOTE

The asterisk (*) designates the ranges that offset compensation is being used.

Verifying 2-wire resistance

Check the normal resistance function by connecting accurate resistance values to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

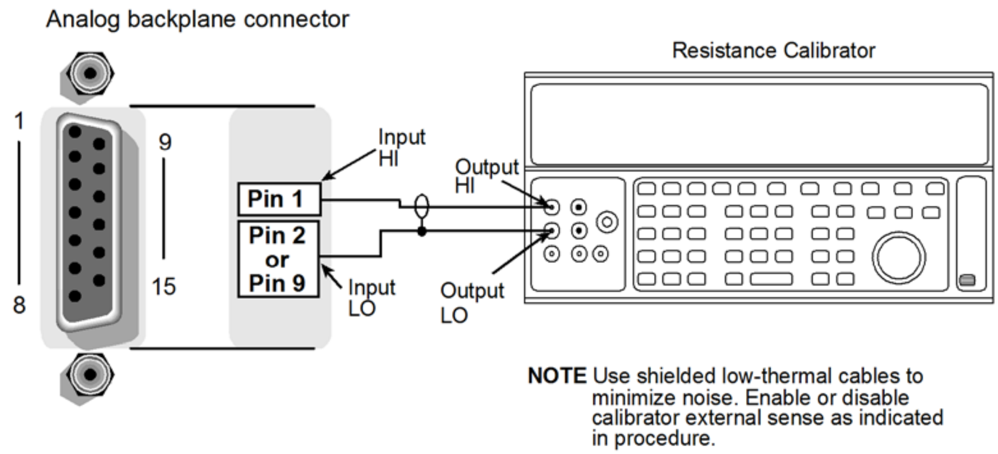
⚠ CAUTION

Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify normal resistance accuracy:

1. Using shielded, Teflon-insulated or equivalent cables in a 2-wire configuration, connect the Model 3706A INPUT and SENSE pins to the calibrator as shown in the "2-wire resistance verification diagram" below.
2. Disable the external sense on the calibrator.
3. Set the Series 3700A to the 2-wire resistance function, set to the proper range.
4. Source a nominal 100 kΩ to 100 MΩ resistance value. Recalculate the limits based on the actual value of the resistor and verify that the reading is within the calculated limits.

Figure 148: 2-wire resistance verification



2-wire resistance verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Calculating resistance reading limits (on page C-4)).

Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify 2W Res 100k Ohm	1.00E+05	1.00E+05	9.999360E+04	1.000064E+05
Verify 2W Res 1M Ohm	1.00E+06	1.00E+06	9.999360E+05	1.000064E+06
Verify 2W Res 10M Ohm	1.00E+07	1.00E+07	9.995900E+06	1.000410E+07
Verify 2W Res 100M Ohm	1.00E+08	1.00E+08	9.979700E+07	1.002030E+08

Verifying dry circuit resistance

Check the dry circuit resistance function by connecting accurate resistance values to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

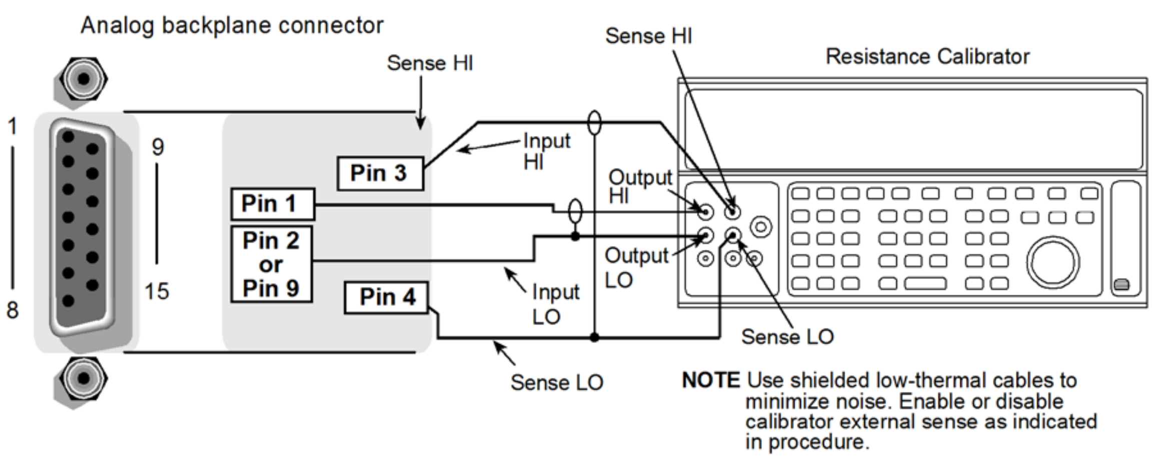
⚠ CAUTION

Do not exceed 300V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify dry circuit resistance accuracy:

1. Using shielded, Teflon-insulated or equivalent cables in a 4-wire configuration, connect the Model 3706A INPUT and SENSE pins to the calibrator as shown for 100 Ω to 10 MΩ ranges.
2. Set the calibrator for 4-wire resistance with external sense on.
3. Select the Model 3706A 4-wire resistance function.
4. Select the SLOW integration rate with the **RATE** key.
5. Enable dry circuit resistance function (see Enabling/disabling dry circuit ohms (see "Enable or disable dry circuit ohms from the front panel" on page 4-60) in the User's manual).
6. Set the Model 3706A for the 100 Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 100 Ω, and 1 kOhm range verification. See Enabling/disabling offset-compensated ohms (on page 4-62) in the User's manual.
7. Recalculate reading limits based on actual calibrator resistance values.
8. Source the nominal full-scale resistance values for the 100 Ω to 2 kΩ ranges summarized in the Dry circuit resistance verification data (on page C-18) table. Verify that the readings are within calculated limits.

Figure 149: Resistance verification



Dry circuit resistance verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Calculating resistance reading limits (on page C-4)).

Description	Range (Ohms)	Test point (Ohms)	Lower limit (Ohms)	Upper limit (Ohms)
Verify Dry Circuit 100 Ohm *	1.00E+02	1.00E+02	9.997800E+01	1.000220E+02
Verify Dry Circuit 1k Ohm	1.00E+03	1.00E+03	9.995200E+02	1.000480E+03
Verify Dry Circuit 2k Ohm	2.00E+03	1.90E+03	1.898320E+03	1.901680E+03

NOTE

The asterisk (*) designates the ranges that offset compensation is being used.

Verifying 1-OHM and 10-OHM resistance ranges

Check the normal resistance function by connecting accurate resistance values to the Model 3706A analog backplane connector and verifying that the displayed readings fall within specified limits.

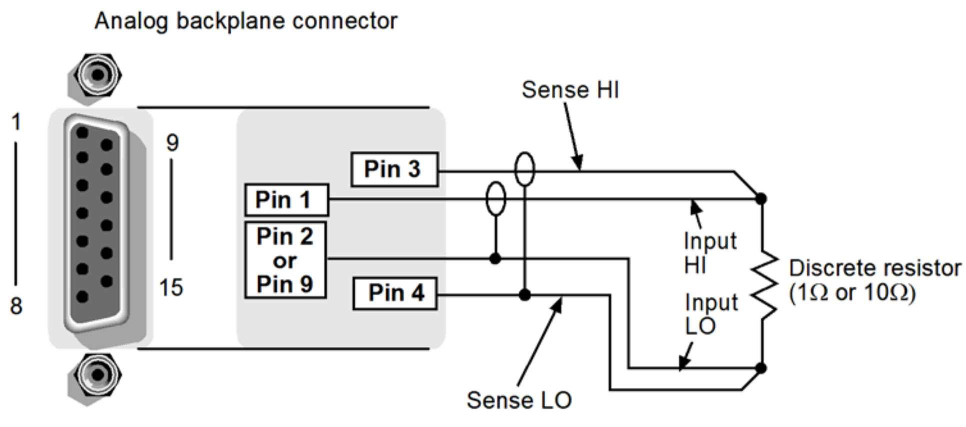
CAUTION

Do not exceed 300 V peak between INPUT HI and INPUT LO because instrument damage may occur.

To verify normal resistance accuracy:

1. Connect the 1 Ω discrete resistor to the Model 3706A input.
2. For the dry circuit test points, enable the dry circuit resistance attribute (DRY+).
3. Select the SLOW integration rate with the **RATE** key.
4. Set the Model 3706A for the 1 Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 1 Ω and 10 Ω range verification.
5. Recalculate reading limits based on actual discrete resistor resistance values.
6. Repeat using the 10 Ω discrete resistor on the 10 Ω range.

Figure 150: Verifying discrete resistance



Discrete resistance verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Calculating resistance reading limits (on page C-4)).

1 Ohm discrete resistor applied				
Description	**Range (Ohms)**	**Test point (Ohms)**	**Lower limit (Ohms)**	**Upper limit (Ohms)**
Verify Res 1 Ohm *	1.00E+00	1.00E+00	9.998600E-01	1.000140E+00
Verify Dry Circuit 1 Ohm *	1.00E+00	1.00E+00	9.998500E-01	1.000150E+00

10 Ohm discrete resistor applied				
Description	**Range (Ohms)**	**Test point (Ohms)**	**Lower limit (Ohms)**	**Upper limit (Ohms)**
Verify Res 10 Ohm *	1.00E+01	1.00E+01	9.999310E+00	1.000069E+01
Verify Dry Circuit 10 Ohm *	1.00E+01	1.00E+01	9.998500E+00	1.000150E+01

NOTE

The asterisk (*) designates the ranges that offset compensation is being used.

Verifying zeros using a 4-wire short

Check the zeros of various test points while the 4-wire is connected to the Model 3706A analog backplane connector and verify that the displayed readings fall within specified limits.

CAUTION

Do not exceed 300 V peak between INPUT HI and INPUT LO because instrument damage may occur.

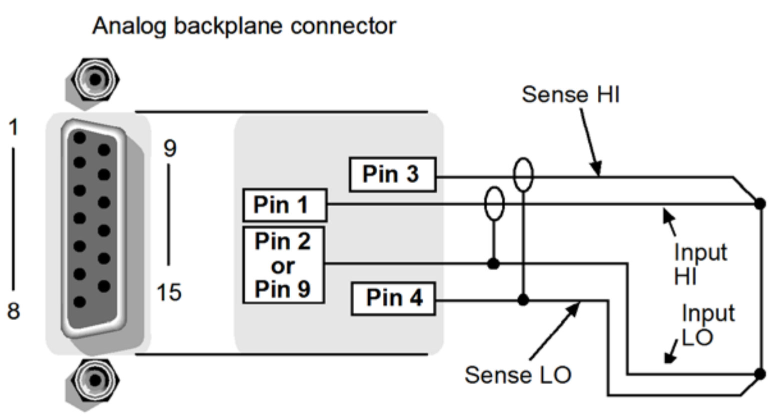
To verify DC voltage and resistance zeros:

1. Select the DC volts function.
2. Set the Model 3706A to the 100 mV range.
3. Connect the 4-wire short to the Model 3706A analog backplane connector and allow to settle for 5 minutes (do not use relative offset).
4. Verify the 100 mV zero is within specification (see the 4-wire short applied verification data (on page C-21) table).
5. Set the Model 3706A to the 1 V range.
6. Allow to settle for 30 seconds (do not use relative offset).
7. Verify the 1 V zero is within specification (see the 4-wire short applied verification data (on page C-21) table).

To verify resistance using the 4-wire short:

1. With the 4-wire short still applied, select the Model 3706A 4-wire resistance function.
2. Select the SLOW integration rate with the **RATE** key.
3. Set the Model 3706A for the 1 Ω range, and make sure the FILTER is on. Enable OC+ (offset-compensated ohms). Use OC+ for 1 Ω and 10 Ω range verification.
4. Verify the 1 Ω range zero is within specification (see the 4-wire short applied verification data (on page C-21) table).
5. Set the Model 3706A for the 10 Ω range (make sure the FILTER is on and OC+ is still enabled).
6. Verify the 10 Ω range zero is within specification (see the 4-wire short applied verification data (on page C-21) table).

Figure 151: 4-wire short diagram



4-wire short applied verification data

Use the following values to verify the performance of the Model 3706A. Actual values depend on published specifications (see Calculating resistance reading limits (on page C-4)).

4-wire short applied				
Description	**Range (V)**	**Test point (V)**	**Lower limit (V)**	**Upper limit (V)**
Verify Zeros 100 mVDC	1.00E-01	0.00E+00	-9.000000E-07	9.000000E-07
Verify Zeros 1 VDC	1.00E+00	0.00E+00	-2.000000E-06	2.000000E-06

Description	**Range (Ohms)**	**Test point (Ohms)**	**Lower limit (Ohms)**	**Upper limit (Ohms)**
Verify Zeros 1 Ohm *	1.00E+00	0.00E+00	-8.000000E-05	8.000000E-05
Verify Zeros 10 Ohm *	1.00E+01	0.00E+00	-9.000000E-05	9.000000E-05

NOTE

The asterisk (*) designates the ranges that offset compensation is being used.

This completes the verification procedure.

Calibration

Overview

Use the procedures in this section to calibrate the Keithley Instruments Model 3706A System Switch/Multimeter.

WARNING

The information in this section is intended for qualified service personnel only. Do not attempt these procedures unless you are qualified to do so.

Some of these procedures may expose you to hazardous voltages, that if contacted, could cause personal injury or death. Use appropriate safety precautions when working with hazardous voltages.

For the plug-in modules, the maximum common-mode voltage (voltage between any plug-in module terminal and chassis ground) is 300 VDC or 300 V_{RMS}. Exceeding this value may cause a breakdown in insulation, creating a shock hazard.

All procedures in this section require accurate equipment calibration to supply precise DC and AC voltages, DC and AC currents, and resistance values. Calibration can be performed any time by an operator using the remote commands sent either over the IEEE-488 bus or Ethernet. DC-only or AC-only calibration may be performed individually, if desired.

Environmental conditions

Conduct the verification procedures in a location that has:

- An ambient temperature of 18 °C to 28 °C (65 °F to 82 °F)
- A relative humidity of less than 80%, unless otherwise noted

Warmup period

NOTE

At the factory, instruments are calibrated without any switch cards installed and all slots are covered with blank slot covers. The slot covers come installed on the instrument when it is shipped.

If it is more convenient to calibrate the instrument with switch cards installed, make sure all channels are open and any empty slots are covered with blank slot covers.

Allow the System Switch/Multimeter to warm up for at least two hours before performing calibration.

If the instrument has been subjected to temperature extremes (those outside the ranges stated in Environmental conditions (on page C-2)), allow extra time for the instrument's internal temperature to stabilize. Typically, you need to allow one extra hour to stabilize an instrument that is 10 °C (18 °F) outside the specified temperature range.

Also, allow the test equipment to warm up for the minimum time specified by the manufacturer.

Line power

The Model 3706A requires a line voltage of 100 V to 240 V (±10%), and a line frequency of 50 Hz or 60 Hz.

NOTE

The instrument automatically senses the line frequency at power-up.

Calibration considerations

When performing calibration procedures:

- Make sure that the equipment is properly warmed up and connected to the appropriate input jacks.
- Make sure the calibrator is in OPERATE mode before you complete each calibration step.
- Always let the source signal settle before calibrating each point.
- If an error occurs during a calibration, the Model 3706A will generate an appropriate error message. See Error summary (on page 9-10) for more information.

WARNING

The input/output terminals of the digital multimeter (DMM) and switch cards are rated for connection to circuits rated Installation Category I only, with transients rated less than 1500V peak. Do not connect the DMM or switch card terminals to CAT II, CAT III, or CAT IV circuits.

Connections of the DMM or switch card terminals to circuits higher than CAT I can cause damage to the equipment or expose the operator to hazardous voltages.

Calibration cycle

Perform calibration at least once a year, or every 90 days to ensure the unit meets the corresponding specifications.

Recommended equipment

The following table lists the recommended equipment and settings you need for DC-only, and AC-only calibration procedures. Alternate equipment may be used, such as a DC transfer standard and characterized resistors, as long as the equipment has specifications at least as good as those listed in the table. In general, equipment uncertainty should be at least four times better (more accurate) than the corresponding Model 3706A specifications.

Manufacturer	Model	Description	Used for:	Uncertainty
Fluke	5700A	Calibrator	All DCV, ACV, DCI, ACI, and Resistance	See NOTE.
N/A	N/A	4-wire short	DCV, resistance zeros	N/A
Agilent	33220A	Function generator	For frequency factory calibration only	See NOTE.

NOTE

Refer to the manufacturer's specifications to calculate the uncertainty, which will vary for each test point.

Calibration

Calibration must be performed by remote control using Ethernet, GPIB, or USB interfaces. No front panel calibration is available. Refer to System connections for more information on communicating with the instrument.

"Factory calibration" refers to additional calibration steps that are only performed once at the factory or when a unit has been repaired by replacing PC boards or components of the boards. The remaining calibration steps can be performed as needed.

The factory calibration steps are:

- **DC Cal Step 0**: A/D MUX Offset, which is performed at the beginning prior to other DC calibration steps
- **Frequency Cal step 17**: 1 V @ 10 Hz and step 18: 1 V @ 1 kHz, which are performed at the end of AC calibration

You can perform individual sections of calibration, but for the instrument to be calibrated properly, all the steps of a section should be performed. For example, DC Cal Step 1: 4-wire short should be done as well as Steps 2 through 5 to properly calibrate DC volts. Other sections are resistance, DC current, AC volts, and AC current. The calibrations must be saved after you have completed all of the steps in order for the adjustments to be permanent.

Before performing a calibration, check the system date of the Model 3706A. This can be done by sending the following command:

```
print(os.date("%x"))
```

If the date is wrong, the date and time need to be reset using the following command:

```
settime(os.time{year = yyyy, month = mm, day = dd, hour =hh, min = mm, sec = ss})
```

Make sure to enter the correct date and time using the 24-hour clock. If the date is incorrect, it will not save the proper date when the calibration is saved. For additional information about this command, see localnode.settime() (see "settime()" on page 8-366).

Remote calibration procedure

To perform calibrations, use the following procedure:

1. Connect the Model 3706A to the IEEE-488 bus of the computer using a shielded IEEE-488 cable, such as the Keithley Instruments Model 7007, over the ethernet, or directly to a computer through the ethernet port using a cross-over cable.
2. Turn on the Model 3706A and allow it to warm up for at least two hours before performing calibrations.
3. Make sure the primary address of the Model 3706A is the same as the address specified in the program that you will be using to send commands (the GPIB default address is 16; the ethernet default port number is 23).

4. Turn the TSP® prompt and errors off and unlock the calibration function by sending the following commands:

```
SEND localnode.prompts=0
SEND localnode.showerrors=0
SEND dmm.reset()
SEND errorqueue.clear()
SEND dmm.calibration.unlock("KI003706")
```

NOTE

When remotely changing the unlock code, send the dmm.calibration.unlock() (on page 8-165) command twice, first with the present code, then with the new code.

5. Check for errors after sending each calibration command by using the following command:

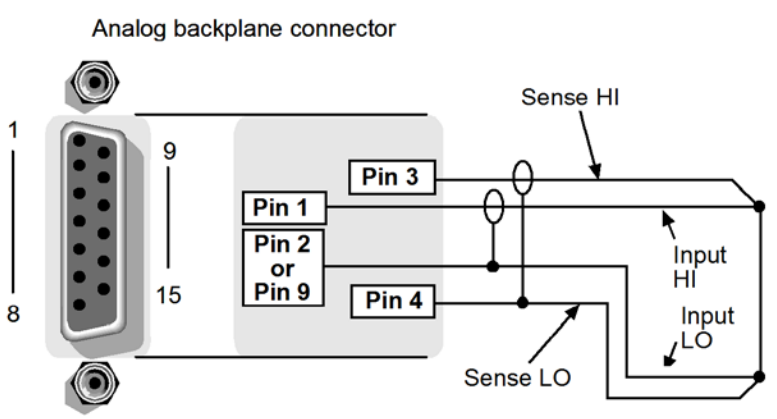
```
SEND print(errorqueue.count)
```

6. Send each calibration command with `print ("done")` appended to allow the program to know when operation is complete. Some calibration steps may take up to five minutes to perform, so the communication time-out setting should be adjusted, because otherwise time-out errors might occur.

DC volts calibration

1. Install the 4-wire short on the analog backplane connector inputs of the Model 3706A.
2. Allow the instrument to settle for five minutes.
3. Perform the following calibration steps (DC Cal Step 0 through Step 5):

Figure 152: 4-wire short diagram



DC adjustment step 0: A/D MUX Offset (factory calibration only)

Send the commands:

```
SEND dmm.calibration.dc(0) print("done")
SEND print(errorqueue.count)
```

DC adjustment step 1: Input four-wire short circuit

1. Allow the instrument to settle for 30 seconds.
2. Send the commands:

```
SEND dmm.calibration.dc(1) print("done")
SEND print(errorqueue.count)
```

DC calibration step 2: Open circuit

1. Remove the four-wire short from the inputs.

NOTE

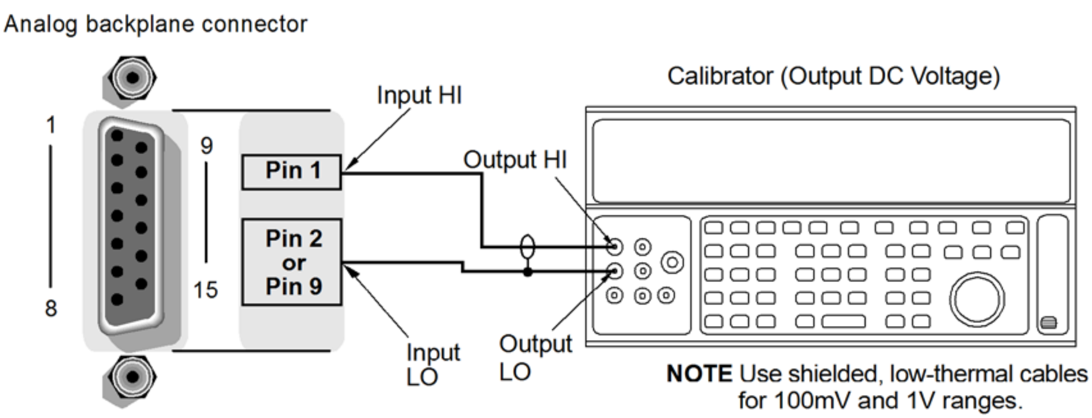
Do not install cables to the inputs (cables will be installed in DC calibration step 3: +10 Volt (see "DC adjustment step 3: +10 V" on page C-26)).

2. Send the commands:

```
SEND dmm.calibration.dc(2) print("done")
SEND print(errorqueue.count)
```

DC adjustment step 3: +10 V

Figure 153: DC voltage calibration



1. Connect a cable between the calibrator and the Model 3706A.
2. Allow the instrument to settle for 30 seconds.
3. Send the command:

```
SEND dmm.range = 10
```

4. Source +10 V.
5. Send the commands:

```
SEND dmm.calibration.dc(3,10) print("done")
SEND print(errorqueue.count)
```

DC adjustment step 4: -10 V

1. Source –10 V.
2. Send the commands:

```
SEND dmm.calibration.dc(4,-10) print("done")
SEND print(errorqueue.count)
```

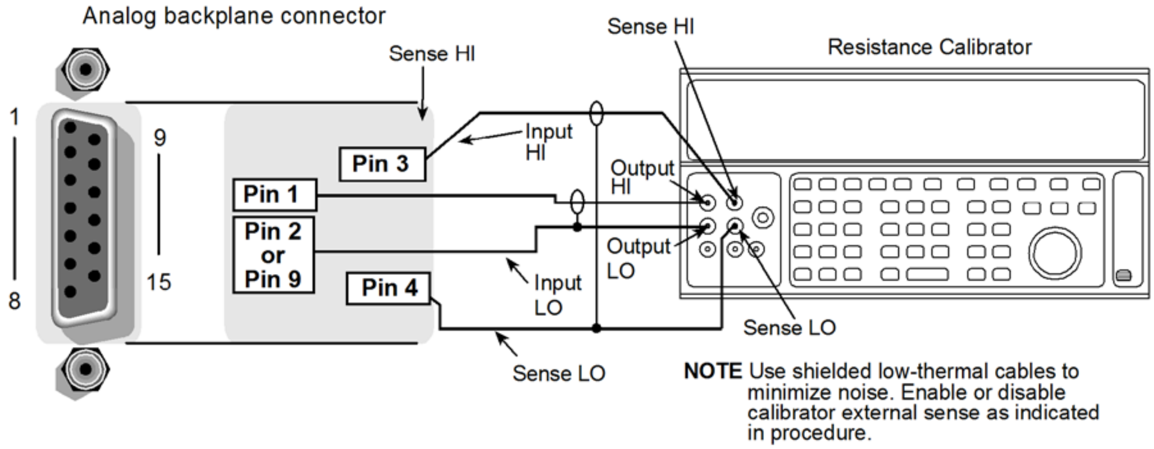
DC adjustment 5: 100 V

1. Send the command:
   `SEND dmm.range = 100`
2. Source 100 V.
3. Send the commands:
   `SEND dmm.calibration.dc(5,100) print("done")`
   `SEND print(errorqueue.count)`

Resistance calibration

Perform the following calibration steps (DC Cal Step 6 through Step 9):

Figure 154: Resistance calibration



DC adjustment step 6: 100 Ohm

1. Send the commands:
   `SEND dmm.func = dmm.FOUR_WIRE_OHMS`
   `SEND dmm.range = 100`
2. Source 100 Ohms.
3. Read the resistor value from the calibrator.
4. Send the command:
   `SEND dmm.calibration.dc(6,(resistor value)) print("done")`

DC adjustment step 7: 10 kOhm

1. Send the command:
   `SEND dmm.range = 10e+3`
2. Source 10 kOhm.
3. Read the resistor value from the calibrator.
4. Send the command:
   `SEND dmm.calibration.dc(7, (resistor value)) print("done")`

DC adjustment step 8: 100 kOhm

1. Send the command:
   `SEND dmm.range = 100e+3`
2. Source 100 kOhm.
3. Read the resistor value from the calibrator.
4. Send the command:
   `SEND dmm.calibration.dc(8, (resistor value)) print("done")`

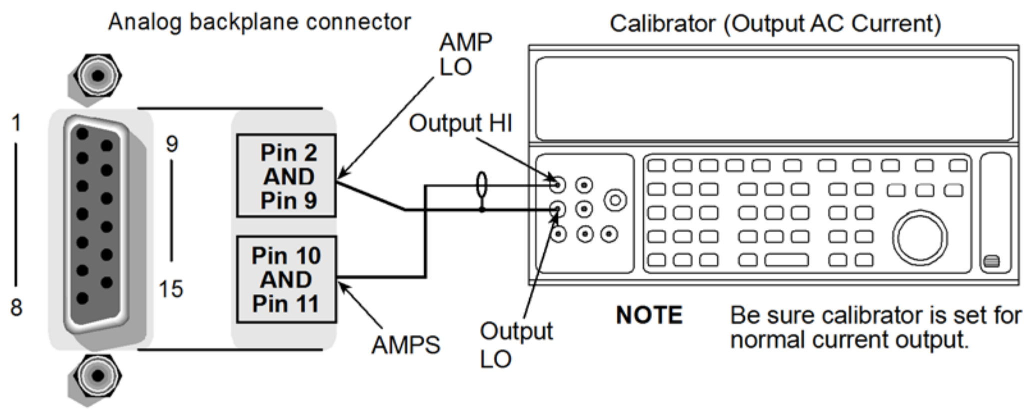
DC adjustment step 9: 1 MOhm

1. Send the command:
   `SEND dmm.range = 1e+6`
2. Source 1 MOhm then read the resistor value from the calibrator.
3. Send the command:
   `SEND dmm.calibration.dc(9, (resistor value)) print("done")`

DC current calibration

Make the connections as shown, then perform the following calibration steps (DC Cal Step 10 through Step 14):

Figure 155: DC current calibration



DC adjustment step 10: 100 µA

1. Send the commands:
   `SEND dmm.func = dmm.DC_CURRENT`
   `SEND dmm.range = 100e-6`
2. Source 100 µA.
3. Send the commands:
   `SEND dmm.calibration.dc(10,.0001) print("done")`

DC adjustment step 11: 1 mA

1. Send the command:
   `SEND dmm.range = 1e-3`
2. Source 1 mA.
3. Send the command:
   `SEND dmm.calibration.dc(11,.001) print("done")`

DC adjustment step 12: 10 mA

1. Send the command:
   `SEND dmm.range = 10e-3`
2. Source 10 mA.
3. Send the command:
   `SEND dmm.calibration.dc(12,.01) print("done")`

DC adjustment step 13: 100 mA

1. Send the command:
   `SEND dmm.range = 100e-3`
2. Source 100 mA.
3. Send the command:
   `SEND dmm.calibration.dc(13,.1) print("done")`

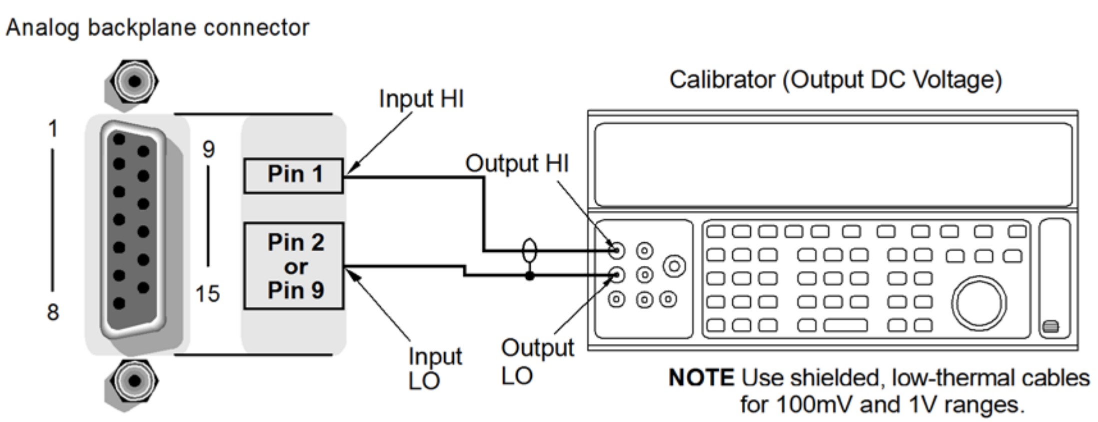
DC adjustment step 14: 1 A

1. Send the command:
   `SEND dmm.range = 1`
2. Source 1 A.
3. Send the command:
   `SEND dmm.calibration.dc(14,1) print("done")`

AC volts calibration

Make the connections as shown below, then perform the following calibration steps (AC Cal Step 1 through Step 10):

Figure 156: AC voltage calibration



AC adjustment step 1: 10 mV at 1 kHz

1. Send the commands:
```
SEND dmm.func = dmm.AC_VOLTS
SEND dmm.range = 10e-3
```
2. Source 10 mV at 1 kHz.
3. Send the command:
```
SEND dmm.calibration.ac(1) print("done")
```

AC adjustment step 2: 100 mV at 1 kHz

1. Send the command:
```
SEND dmm.range = 100e-3
```
2. Source 100 mV at 1 kHz.
3. Send the command:
```
SEND dmm.calibration.ac(2) print("done")
```

AC adjustment step 3: 100 mV at 50 kHz

1. Source 100 mV at 50 kHz.
2. Send the command:
```
SEND dmm.calibration.ac(3) print("done")
```

AC adjustment step 4: 1 V at 1 kHz

1. Send the command:
```
SEND dmm.range = 1
```
2. Source 1 V at 1 kHz.
3. Send the command:
```
SEND dmm.calibration.ac(4) print("done")
```

AC adjustment step 5: 1 V at 50 kHz

1. Source 1 V at 50 kHz.
2. Send the command:
```
SEND dmm.calibration.ac(5) print("done")
```

AC adjustment step 6: 10 V at 1 kHz

1. Send the command:
```
SEND dmm.range = 10
```
2. Source 10 V at 1 kHz.
3. Send the command:
```
SEND dmm.calibration.ac(6) print("done")
```

AC adjustment step 7: 10 V at 50 kHz

1. Source 10 V at 50 kHz.
2. Send the command:
```
SEND dmm.calibration.ac(7) print("done")
```

AC adjustment step 8: 100 V at 1 kHz

1. Send the command:
   `SEND dmm.range = 100`
2. Source 100 V at 1 kHz.
3. Send the command:
   `SEND dmm.calibration.ac(8) print("done")`

AC adjustment step 9: 100 V at 50 kHz

1. Source 100 V at 50 kHz.
2. Send the command:
   `SEND dmm.calibration.ac(9) print("done")`

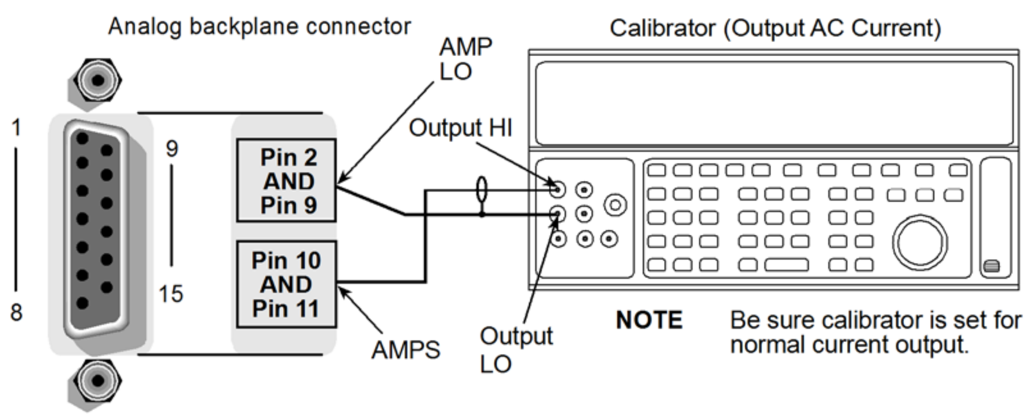
AC adjustment step 10: 300 V at 1 kHz

1. Send the command:
   `SEND dmm.range = 300`
2. Source 300 V at 1 kHz
3. Send the command:
   `SEND dmm.calibration.ac(10) print("done")`

AC current calibration

Make the connections as shown, then perform the calibration steps (AC calibration step 11 through step 16).

Figure 157: AC current calibration 1mA to 1A range



AC adjustment step 11: 100 µA at 1 kHz

1. Send the commands:
   `SEND dmm.func = dmm.AC_CURRENT`
   `SEND dmm.range = 100e-6`
2. Source 100 µA at 1 kHz.
3. Send the command:
   `SEND dmm.calibration.ac(11) print("done")`

AC adjustment step 12: 1 mA at 1 kHz

1. Send the following command:
   ```
   SEND dmm.range = 1e-3
   ```
2. Source 1 mA at 1 kHz.
3. Send the following command:
   ```
   SEND dmm.calibration.ac(12) print("done")
   ```

AC adjustment step 13: 10 mA at 1 kHz

1. Send the command:
   ```
   SEND dmm.range = 10e-3
   ```
2. Source 10 mA at 1 kHz.
3. Send the command:
   ```
   SEND dmm.calibration.ac(13) print("done")
   ```

AC adjustment step 14: 100 mA at 1 kHz

1. Send the command:
   ```
   SEND dmm.range = 100e-3
   ```
2. Source 100 mA at 1 kHz.
3. Send the command:
   ```
   SEND dmm.calibration.ac(14) print("done")
   ```

AC adjustment step 15: 1 A at 1 kHz

1. Send the command:
   ```
   SEND dmm.range = 1
   ```
2. Source 1 A at 1 kHz.
3. Send the command:
   ```
   SEND dmm.calibration.ac(15) print("done")
   ```

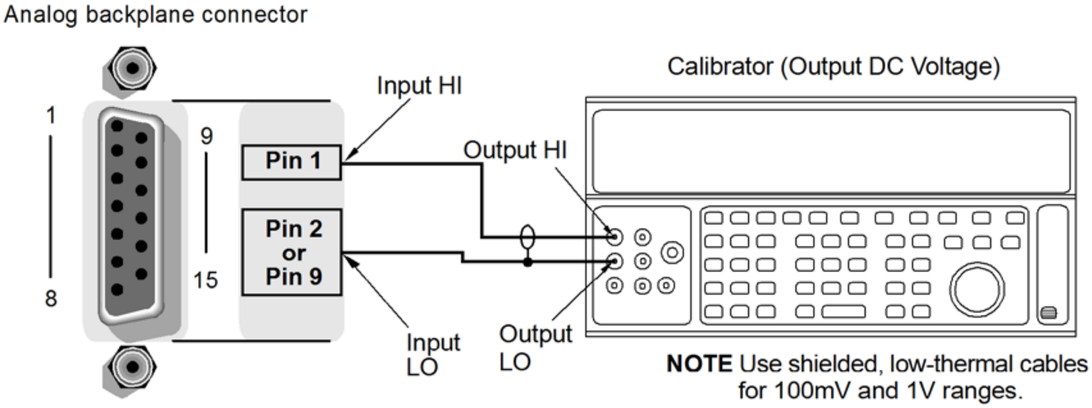
AC adjustment step 16: 2 A at 1 kHz

1. Send the command:
   ```
   SEND dmm.range = 2
   ```
2. Source 2 A at 1 kHz.
3. Send the command:
   ```
   SEND dmm.calibration.ac(16) print("done")
   ```

Frequency calibration

Make the connections as shown below, then perform the following calibration steps (AC Cal Step 17 and Step 18):

Figure 158: Low frequency calibration

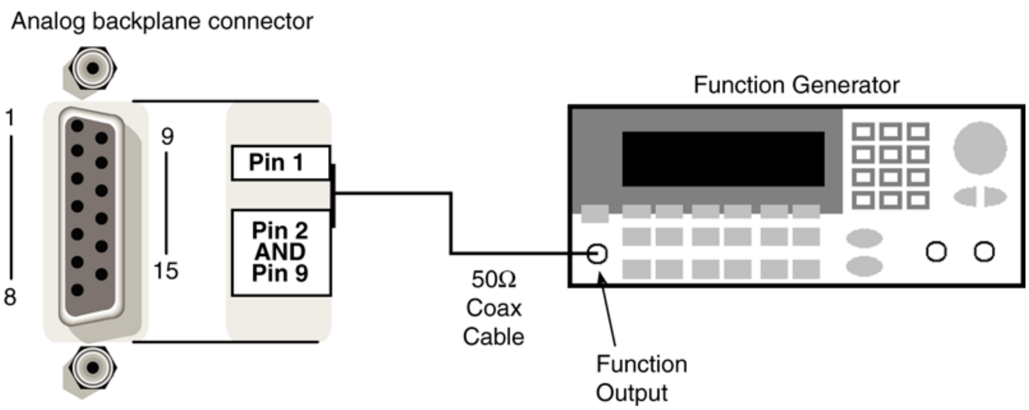


AC adjustment step 17: 1 V at 10 Hz (factory calibration only)

1. Send the commands:
   ```
   SEND dmm.func = dmm.AC_VOLTS
   SEND dmm.range = 1
   ```
2. Source 1 V at 10 Hz.
3. Send the command:
   ```
   SEND dmm.calibration.ac(17,1) print("done")
   ```

AC adjustment step 18: 1 V at 1 kHz (factory calibration only)

Figure 159: Frequency verification



1. Source 1 V at 1 kHz
2. Send the command:
   ```
   SEND dmm.calibration.ac(18,1000) print("done")
   ```

Save calibrations

Program today's date, calibration due date, and serial number, and save the calibration constants in EEPROM (electrically erasable programmable read-only memory) by sending the following commands:

```
dmm.adjustment.date=os.time()
dmm.calibration.save()
dmm.calibration.verifydate=dmm.adjustment.date
dmm.calibration.lock()
dmm.reset()
```

NOTE

Calibrations are complete after they have been saved and locked.

Appendix D

Status model

In this appendix:

Overview

Each Keithley Instruments Series 3700A provides a number of status registers and queues that are collectively referred to as the status model. Through manipulation and monitoring of these registers and queues, you can view and control various instrument events. You can include commands in your test program that can determine if a service request (SRQ) event has occurred and the cause of the event.

The heart of the status model is the Status Byte Register. All status model registers and queues flow into the Status Byte Register.

The entire status model is illustrated in the Status model diagrams.

Status Byte Register

The Status Byte Register receives summary bits from the other status register sets and queues, and also from itself (which sets the Master Summary Status, or MSS, bit). For details, see Status Byte Register (on page D-17).

Status register set contents

Typically, a status register set contains the following registers:

- **Condition** (`.condition`): A read-only register that is constantly updated to reflect the present operating conditions of the instrument.
- **Enable Register** (`.enable`): A read-write register that allows a summary bit to be set when an enabled event occurs.
- **Event Register** (`.event`): A read-only register that sets a bit to 1 when the applicable event occurs. If the enable register bit for that event is also set, the summary bit of the register will set to 1.
- **Negative Transition Register (NTR)** (`.ntr`): When a bit is set in this read-write register, it enables a 1 to 0 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.
- **Positive Transition Register (PTR)** (`.ptr`): When a bit is set in this read-write register, it enables a 0 to 1 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.

An event is represented by a condition register bit changing from a 1 to 0 or 0 to 1. When an event occurs and the appropriate NTR or PTR bit is set, the corresponding event register bit is set to 1. The event bit remains latched to 1 until the event register is read or the status model is reset. When an event register bit is set and its corresponding enable bit is set, the summary bit of the register is set to 1. This, in turn, sets a bit in a higher-level condition register, potentially cascading to the associated summary bit of the Status Byte Register.

Summary bit

The summary bit of each register is either set (1) or clear (0). A set summary bit indicates that one (or more) of the enabled events in that register has occurred.

Queues

The Model 3706A uses queues to store messages. The queues include:

- Output queue: Holds response messages.
- Error queue: Holds error and status messages.

When a queue contains data, it sets the condition bit for that queue in one of the registers. The condition bits are:

- Command queue: CAV in the Operation Status Remote Summary Register
- Output queue: MAV in the Status Byte Register
- Error queue: EAV in the Status Byte Register

The CAV, MAV, and EAV bits in the registers are cleared when the queue is empty. Queues empty when:

- Commands are executed
- Errors are read from the error queue
- Response messages are read from the instrument

All Model 3706A queues are first-in, first-out (FIFO).

The Status model diagrams shows how the queues are structured with the other registers.

Output queue

When the instrument is in the remote state, the output queue holds data that pertains to the normal operation of the instrument. For example, when a `print()` command is sent, the response message is placed in the output queue.

When data is placed in the output queue, the Message Available (MAV) bit in the status byte register is set. A response message is cleared from the output queue when it is read. The output queue is considered cleared when it is empty. An empty output queue clears the MAV bit in the status byte register.

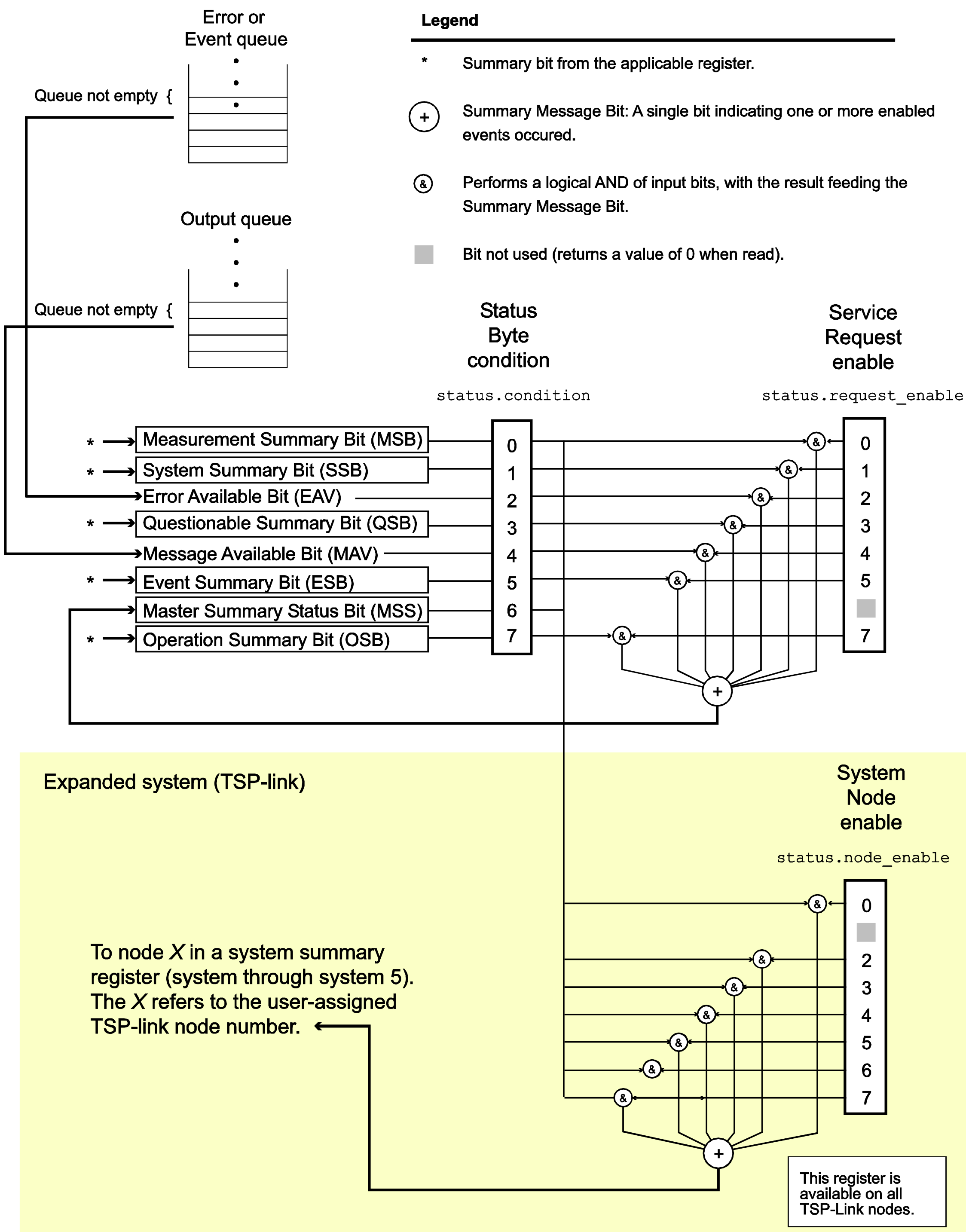
A message is read from the output queue by addressing the instrument to talk.

Status model diagrams

The register sets (and queues) monitor various instrument events. When an enabled event occurs in one of the five registers, it sets the associated summary bit in the Status Byte register. When a summary bit of the Status Byte is set and its corresponding enable bit is set (as programmed using `status.request_enable`), the MSS bit will set to indicate that an SRQ has occurred. View the master summary bit using `status.condition` attribute. In an expanded system (TSP-link), setting the `status.node_enable` attribute allows the System registers to be shared by all nodes in the TSP-Link system. The following figures and topics illustrate the relationships of the individual registers and queues with the Status Byte register.

Status Byte Register overview

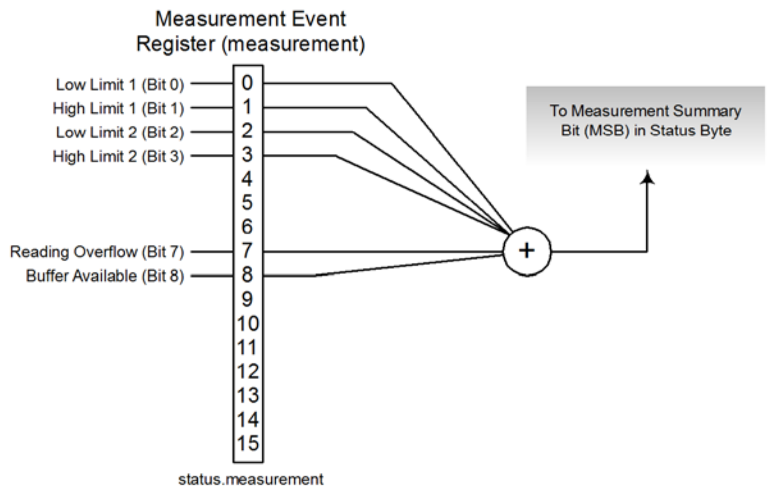
Figure 160: Status Byte register



Measurement summary bit (Measurement event register)

The summary bit of the measurement event register provides enabled summary information to Bit B0 (MSB) of the status byte.

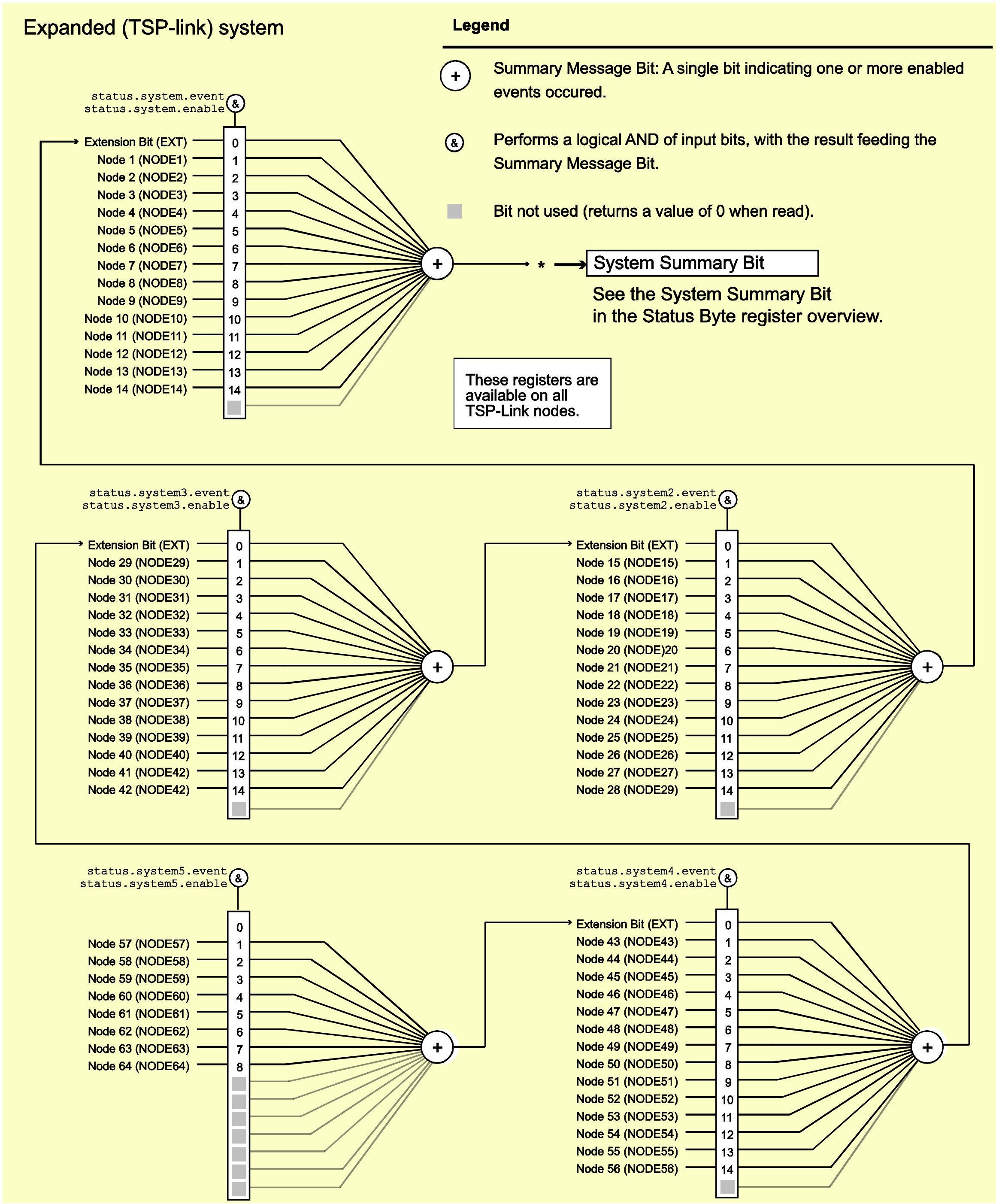
Figure 161: Measurement event register



System summary bit (System register)

The summary bit of the system register provides enabled summary information to Bit B1 (SSB) of the status byte.

Figure 162: System summary bit (System register)



As shown above, there are five register sets associated with System Event Status. These registers summarize system status for various nodes connected to the TSP-Link. Note that all nodes on the TSP-Link share a copy of the system summary registers once the TSP-Link has been initialized. This feature allows all nodes to access the status models of other nodes, including SRQ.

In a TSP-Link system, the status model can be configured such that a status event in any node in the system can set the RQS (Request for Service) bit of the Master Node Status Byte. See TSP-Link system status (on page D-22) for details on using the status model in a TSP-Link system.

Attributes are summarized in status.system.* (on page 8-411), status.system2.* (on page 8-413), status.system3.* (on page 8-415), status.system4.* (on page 8-417), and status.system5.* (on page 8-419).

For example, any of the following commands will set the EXT enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = status.system.EXTENSION_BIT
status.system.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Reading registers (on page D-16). For example, the following command will read the system enable register:

```
print(status.system.enable)
```

The bits used in the system register sets are described as follows:

- **Bit B0, Extension Bit (EXT):** Set bit indicates that an extension bit from another system status register is set.
- **Bits B1-B14* NODE*N*:** Indicates a bit on TSP-Link node n has been set (*N* = 1 to 64).
- **Bits B15:** Not used.

`*status.system5` does not use bits B9 through B15.

Refer to the following table for available *N* values:

Command	***N* value**
`status.system.*`	1 to 14
`status.system2.*`	15 to 28
`status.system3.*`	29 to 42
`status.system4.*`	43 to 56
`status.system5.*`	57 to 64

Standard Event Register

The bits used in the Standard Event Register are described as follows:

- **Bit B0, Operation Complete (OPC):** Set bit indicates that all pending selected device operations are completed and the Series 3700A instrument is ready to accept new commands. The bit is set in response to an *OPC command. The `opc()` function can be used in place of the *OPC command. See Common commands for details on the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, Query Error (QYE):** Set bit indicates that you attempted to read data from an empty output queue.
- **Bit B3, Device-Dependent Error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE):** Set bit indicates that the Series 3700A instrument detected an error while trying to execute a command.
- **Bit B5, Command Error (CME):** Set bit indicates that a command error has occurred. Command errors include:
  - IEEE Std 488.2 syntax error: The Series 3700A instrument received a message that does not follow the defined syntax of IEEE Std 488.2.
  - Semantic error: Series 3700A instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented.
  - The instrument received a Group Execute Trigger (GET) inside a program message.
- **Bit B6, User Request (URQ):** Set bit indicates that the LOCAL key on the Series 3700A instrument front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the Series 3700A instrument has been turned off and turned back on since the last time this register was read.

Commands to program and read the register are summarized below and also in the Status function summary (on page D-14) table.

Standard event commands

Command	Description
`*ESR?` or `print(status.standard.event)`	Read Standard Event Status Register.
`*ESE <mask>` or `status.standard.enable = <mask>`	Program the Event Status Enable Register: `<mask>` = 0 to 255 See Status register set contents (on page D-2).
`*ESE?` or `print(status.standard.enable)`	Read Event Status Enable Register.

Error available bit (Error or Event queue)

The summary bit of the Error or Event queue provides enabled summary information to Bit B2 (EAV) of the status byte.

The Error Available Bit (EAV) is set when a message defining an error (or status) is placed in the Error or Event queue. The Error or Event queue is one of the two System Switch/Multimeter queues associated with the status model. The other queue sets the Message available bit (Output queue)). Both queues are first-in, first-out (FIFO) queues. The Error queue holds error and status messages. The status model shows how these queues are structured with regard to the other registers.

The following sequence outlines typical events associated with this queue:

1. When an error or status event occurs, a message defining the error (or status) is placed in the Error queue.
2. The Error Available (EAV) bit in the Status Byte Register is set.
3. Through programming, the error (or status) message is read. This clears the error (or status) from the Error Queue. The Error queue is considered cleared when it is empty.
4. An empty Error queue clears the EAV bit in the Status Byte Register.

The commands to control the Error queue are listed below. When you read a single message in the Error queue, the oldest message is read and then removed from the queue. On power-up, the Error queue is initially empty. If there are problems detected during power-on, entries will be placed in the queue. If no problems are detected, the error number `0` and "`No Error`" will be returned.

Error queue command	**Description**
`errorqueue.clear()`	Clear error queue of all errors.
`errorqueue.count`	Number of messages in the error/event queue.
`errorqueue.next()`	Request error message.

Messages in the Error queue include a code number, message text, severity, and TSP-Link node number. For example, the following commands request the next complete error information from the error queue and displays the code, message, severity and node of the next error:

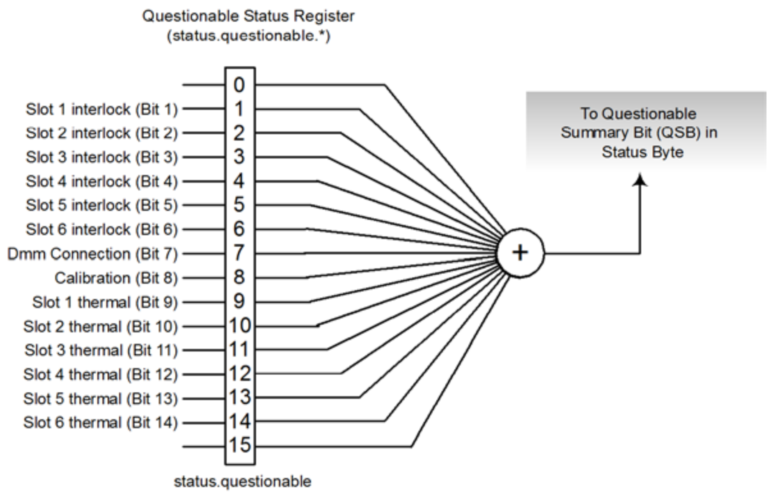
```
errorcode, message, severity, errornode = errorqueue.next()
print(errorcode, message, severity, errornode)
```

The error messages, as well as error numbers, are listed in the Error summary list.

Questionable summary bit (Questionable event register)

The summary bit of the questionable event register provides enabled summary information to Bit B3 (QSB) of the status byte.

Figure 163: Questionable event register



As shown above, there is only one register set associated with the questionable status. Attributes are summarized in status.questionable.* (on page 8-403). Keep in mind that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers (on page D-15).

For example, any of the following statements will set the thermal aspect enable bit of a card in slot 1:

```
status.questionable.enable = status.questionable.S1THR
status.questionable.enable = status.questionable.SLOT1_THERMAL
status.questionable.enable = 512
```

The following command will request the questionable enable register value in numeric form:

```
print(status.questionable.enable)
```

The bits used in this register set are described as follows:

- **SxTHR:** Set bit indicates the thermal aspect of the card in slot x is in question, where x = 1 to 6.

Message available bit (Output queue)

The summary bit of the output queue provides enabled summary information to Bit B4 (MAV) of the status byte.

The Message Available Bit (MAV) is set when the Output queue holds data that pertains to the normal operation of the instrument. The Output queue is one of the two System Switch/Multimeter queues associated with the status model. The other queue sets the Error Available Bit (Error or Event queue) (on page D-9). Both queues are first-in, first-out (FIFO) queues. The Status Byte Register overview (on page D-4) shows how these queues are structured with regard to the other registers.

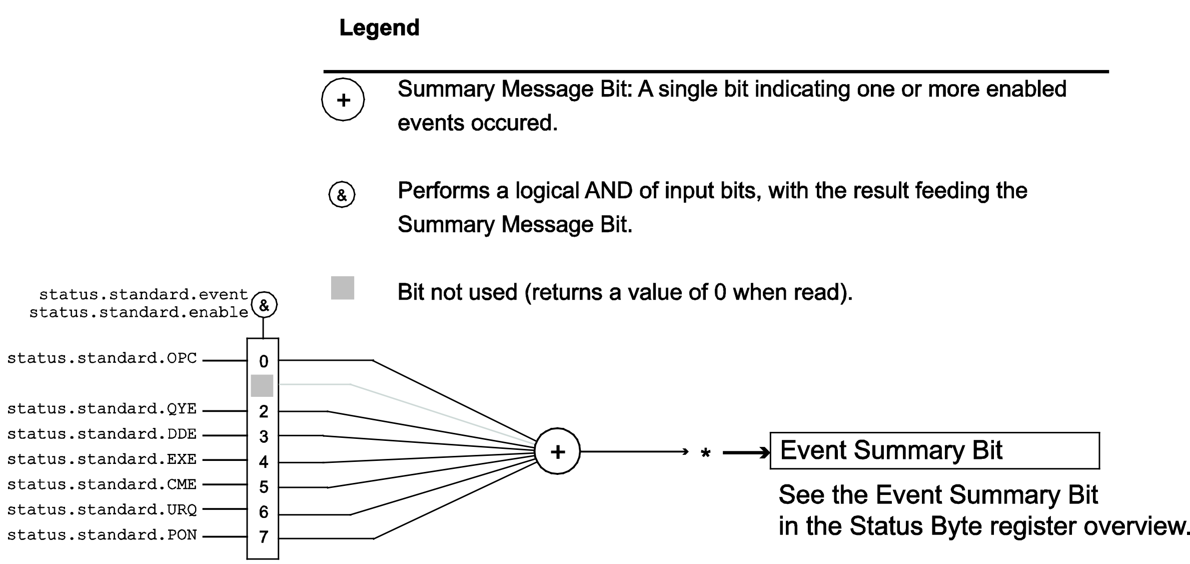
As an example, when a print command is sent, the response message is placed in the Output queue. When data is placed in the Output queue, the Message Available (MAV) bit in the Status Byte Register sets. A response message is cleared from the Output queue when it is read. The Output queue is considered cleared when it is empty. An empty Output queue clears the MAV bit in the Status Byte Register.

A message is read from the Output queue by addressing the System Switch/Multimeter to talk.

Event summary bit (ESB register)

The summary bit of the Standard event register provides enabled summary information to Bit B5 (OSB) of the status byte.

Figure 164: Event summary bit (Standard event register)



As shown above, there is only one register set associated with the event status register. Attributes are summarized in status.standard.* (on page 8-409). Keep in mind that bits can also be set by using numeric parameter values. For details, see Programming enable and transition registers (on page D-15).

For example, any of the following statements will set the operation complete enable bit:

```
standardRegister = status.standard.OPC
status.questionable.enable = status.standard.OPERATION_COMPLETE
status.questionable.enable = 1
```

The bits used in this register set are described as follows:

- **Bit B0, Operation Complete (OPC):** Set bit indicates that all pending selected device operations are completed and the instrument is ready to accept new commands. The bit is set in response to an *OPC command. The remote command `opc()` can be used in place of the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, Query Error (QYE):** Set bit indicates that you attempted to read data from an empty Output queue.
- **Bit B3, Device-Dependent Error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE):** Set bit indicates that the instrument detected an error while trying to execute a command.
- **Bit B5, Command Error (CME):** Set bit indicates that a command error has occurred. Command errors include:
  - IEEE-488.2 syntax error: The instrument received a message that does not follow the defined syntax of the IEEE-488.2 standard.
  - Semantic error: instrument received a command that was misspelled or received an optional IEEE-488.2 command that is not implemented.
  - GET error: The instrument received a Group Execute Trigger (GET) inside a program message.
- **Bit B6, User Request (URQ):** Set bit indicates that the LOCAL key on the instrument front panel was pressed.
- **Bit B7, Power ON (PON):** Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read.

Master summary status bit (MSS bit register)

The master summary status bit provides summary information to Bit B6 (MSS) of the status byte. Although this bit is always enabled for the status byte, it has to be enabled (using `status.node_enable)` if needed in an expanded system (TSP-link).

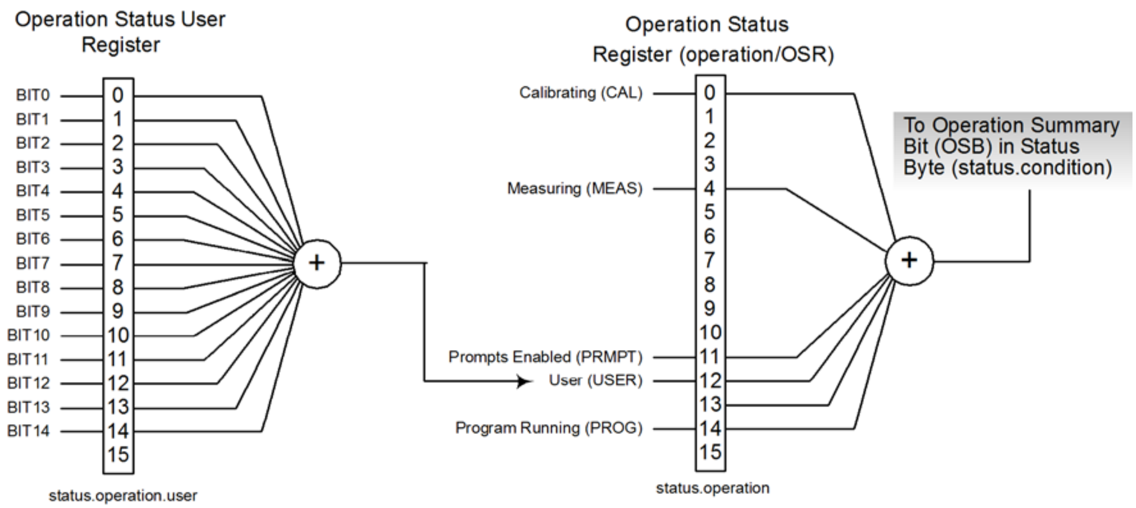
The Master Summary Status Bit (MSS) is set when an enabled summary bit of the Status Byte Register is set. This bit (B6) may also be interpreted as a Request Service (RQS) bit. Depending on how it is used, Bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit.

When using the GPIB serial poll sequence of the System Switch/Multimeter to obtain the status byte (serial poll byte), B6 is the RQS bit. See Serial polling and SRQ (on page D-19) for details on using the serial poll sequence. For common and script commands (Status Byte Register), B6 is the MSS (Message Summary Status) bit. The serial poll, although automatically resetting the RQS bit, does not clear MSS. The MSS remains set until all Status Byte summary bits are reset.

Operation summary bit (Operation event register)

The summary bit of the operation event register provides enabled summary information to Bit B7 (OSB) of the status byte.

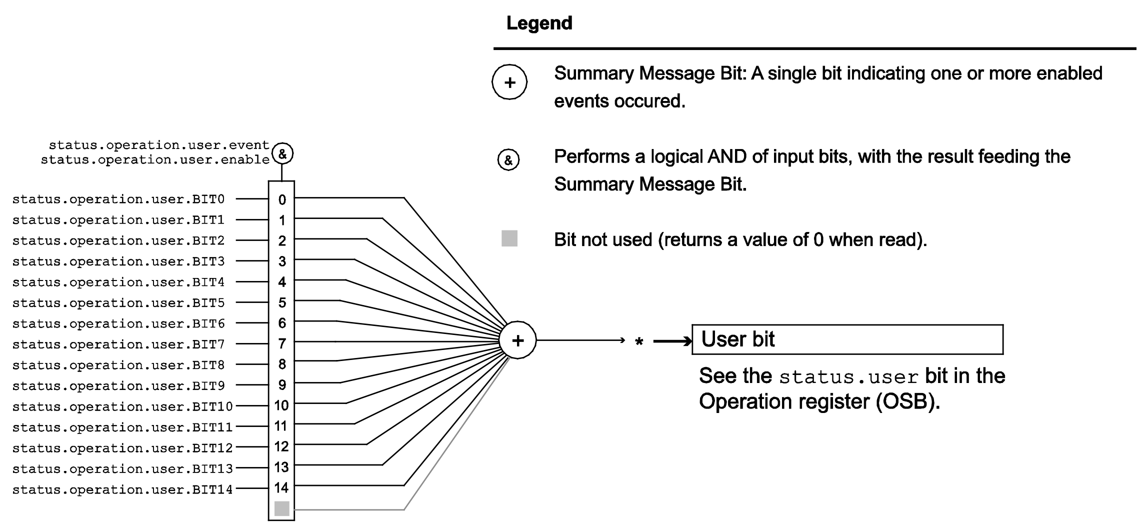
Figure 165: Operation event registers



Operation user bit (Operation user register)

The summary bit of the operation user register provides the user bit (User) (Bit B12) to the operation status register. In turn, the summary bit of the operation status register will provide the operation summary bit (OSB) (Bit B7) to the status byte.

Figure 166: Operation user summary bit (Operation user register)



The bits used in this register set are described as follows:

- **Bits B0-B14:** `status.operation.user.BIT0` through `status.operation.user.BIT14`
- **Bits B15:** Not used.

Status function summary

The following functions and attributes control and read the various registers. Additional information is included in the command listings for the various register sets.

Status function summary

Type	Function or attribute*
System summary	status.condition (on page 8-391) status.node_event (on page 8-398) status.node_enable (on page 8-396) status.request_event (on page 8-407) status.request_enable (on page 8-405) status.reset() (on page 8-409)
Measurement event	status.measurement.* (on page 8-393)
Operation event	status.operation.* (on page 8-399) status.operation.user.* (on page 8-401)
Questionable event	status.questionable.* (on page 8-403)
Standard event	status.standard.* (on page 8-409)
System events	status.system.* (on page 8-411) status.system2.* (on page 8-413) status.system3.* (on page 8-415) status.system4.* (on page 8-417) status.system5.* (on page 8-419)

* Note that the asterisk (`*`) at the end of a command represents one of the following: `.ntr`, `.ptr`, `.enable`, `.event`, or `.condition`.

Clearing registers

Commands to reset the status registers are listed in the table below.

In addition to these commands, you can reset the enable registers and the NTR to 0. To do this, send the individual command to program the register with a 0 as its parameter value. The PTR registers can be reset to their defaults by programming them with all bits on. Note that the event registers are not programmable but can be cleared by reading them.

Commands to reset registers	
Command	**Description**
To reset registers:	
`*CLS`	Reset bits of the event and NTR registers to 0 and set all PTR register bits on. Also clears the output queue.
`status.reset()`	Reset bits of the event and NTR registers to 0 and set all PTR register bits on.

Startup state

When the System Switch/Multimeter is turned on, various register status elements are set as follows:

- The power on (PON) bit in the `status.operation.condition` register is set.
- Other bits are set appropriately based on the instrument's power-on configuration.
- All enable registers (.enable) are set to 0.
- All negative transition registers (.ntr) are set to 0.
- All used positive transition registers (.ptr) bits are set to 1.
- The two queues are empty.

Programming and reading registers

Programming enable and transition registers

The only registers that you can program are the enable and transition registers. All other registers in the status structure are read-only registers. The following explains how to determine the parameter values for the various commands used to program enable registers. The actual commands are summarized in Status function summary (on page D-14).

A command to program an event enable or transition register is sent with a parameter value that determines the desired state (0 or 1) of each bit in the appropriate register. The bit positions of the register (see the following figure) indicate the binary parameter value and decimal equivalent. To program one of the registers, send the decimal value for the bits to be set. The registers are discussed further in Enable and transition registers (on page D-21).

Figure 167: 16-bit status register

Bit position	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	128 (2^7)	64 (2^6)	32 (2^5)	16 (2^4)	8 (2^3)	4 (2^2)	2 (2^1)	1 (2^0)

A. Bits 0 through 7

Bit position	B15	B14	B13	B12	B11	B10	B9	B8
Binary alue	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal Weights	32768 (2^{15})	16384 (2^{14})	8192 (2^{13})	4096 (2^{12})	2048 (2^{11})	1024 (2^{10})	512 (2^9)	256 (2^8)

B. Bits 8 through 15

When using a numeric parameter, registers are programmed by including the appropriate <mask> value. For example:

```
*ese 1169
status.standard.enable = 1169
```

To convert from decimal to binary, use the information shown in the above figure. For example, to set bits B0, B4, B7, and B10, a decimal value of 1169 would be used for the mask parameter (1169 = 1 + 16 + 128 + 1024).

Reading registers

Any register in the status structure can be read either by sending the common command query (where applicable), or by including the script command for that register in either the `print()` or `print(tostring())` command. The `print()` command outputs a numeric value; the `print(tostring())` command outputs the string equivalent. For example, any of the following commands requests the Service Request Enable Register value:

```
*SRE?
print(tostring(status.request_enable))
print(status.request_enable)
```

The response message will be a decimal value that indicates which bits in the register are set. That value can be converted to its binary equivalent using the information in Programming enable and transition registers (on page D-15). For example, for a decimal value of 37 (binary value of 100101), bits B5, B2, and B0 are set.

Register programming example

The command sequence below programs the instrument to generate a service request (SRQ) and set the system summary bit in all TSP-Link nodes when the current limit on channel A is exceeded.

```
-- Clear all registers.
status.reset()

-- Enable SLOT1_THERMAL bit in questionable register.
status.questionable.enable = status.questionable.SLOT1_THERMAL

-- Set the system summary node QSB enable bit.
status.node_enable = status.QSB

-- Set the QSB bit of the service request enable register.
status.request_enable = status.QSB
```

Status byte and service request (SRQ)

Service requests (SRQs) allow an instrument to indicate that it needs attention or that some event has occurred. When the controller receives an SRQ, it allows the controller to interrupt tasks to perform other tasks in order to address the request for service.

For example, you might program your instrument to send an SRQ when:

- All instrument operations are complete
- An instrument error occurs
- A specific operation has occurred

Two 8-bit registers control service requests, the Status Byte Register and the Service Request Enable Register. The Status Byte Register (on page D-17) topic describes the structure of these registers.

Service requests affect GPIB, USB, and VXI-11 connections. On a GPIB connection, the SRQ line is asserted. On a VXI-11 or USB connection, an SRQ event is generated.

Service Request Enable Register

The Service Request Enable Register controls the generation of a service request. 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 D-17) topic, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic “1” output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

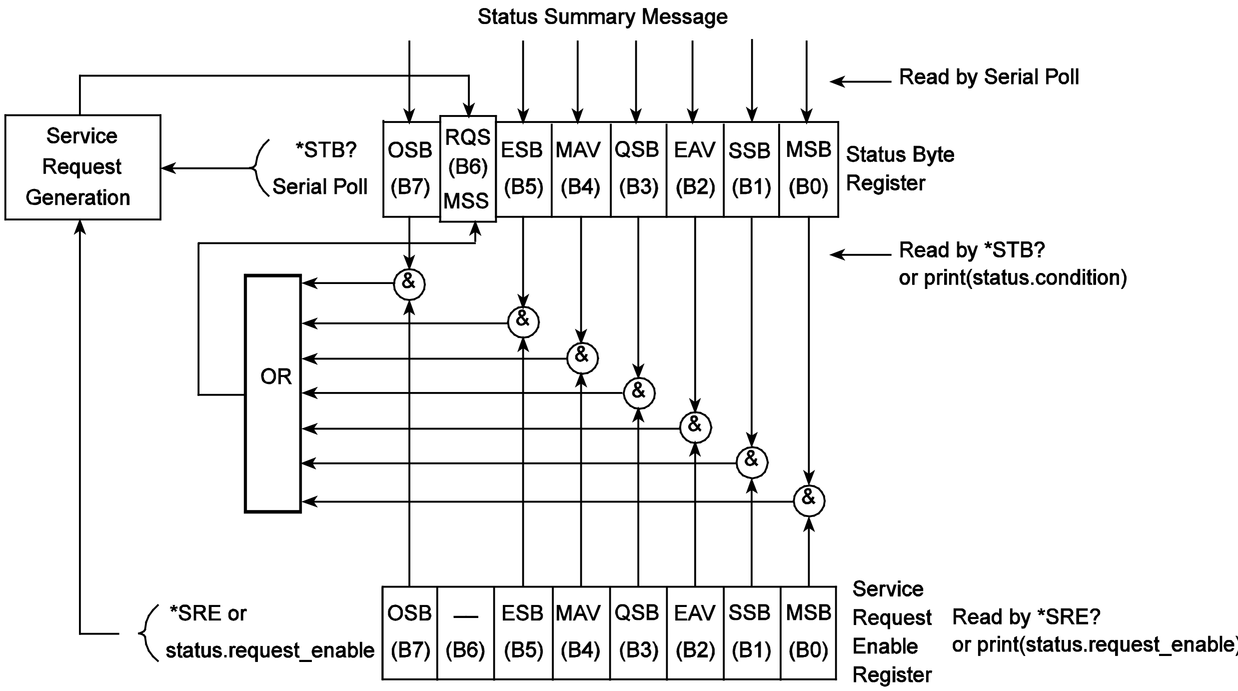
The individual bits of the Service Request Enable Register can be set or cleared by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). The commands to program and read the SRQ Enable Register are listed in Status byte and service request commands (on page D-21).

Status Byte Register

The summary messages from the status registers and queues are used to set or clear the appropriate bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. These summary bits do not latch, and their states (0 or 1) are dependent upon the summary messages (0 or 1). For example, if the Standard Event Register is read, its register will clear. As a result, its summary message will reset to 0, which will then reset the ESB bit in the Status Byte Register.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status, or MSS, bit.

Figure 168: Status byte and service request (SRQ)



OSB = Operation Summary Bit
MSS = Master Summary Status
RQS = Request for Service
ESB = Event Summary Bit
Mav = Message Available
QSB = Questionable Summary Bit
EAV = Error Available
SSB = System Summary Bit
MSB = Measurement Summary Bit
& = Logical AND
OR = Logical OR

The bits of the Status Byte Register are described as follows:

- **Bit B0, Measurement Summary Bit (MSB):** Set summary bit indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB):** Set summary bit indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV):** Set bit indicates that an error or status message is present in the error queue.
- **Bit B3, Questionable Summary Bit (QSB):** Set summary bit indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV):** Set bit indicates that a response message is present in the output queue.
- **Bit B5, Event Summary Bit (ESB):** Set summary bit indicates that an enabled standard event has occurred.
- **Bit B6, Request Service (RQS)/Master Summary Status (MSS):** Set bit indicates that an enabled summary bit of the Status Byte Register is set. Depending on how it is used, bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
  - When using the GPIB, USB, or VXI-11 serial poll sequence of the Series 3700A to obtain the status byte (serial poll byte), B6 is the RQS bit. See Serial polling and SRQ (on page D-19) for details on using the serial poll sequence.
  - When using the *STB? common command or status.condition Status byte and service request commands (on page D-21) to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB):** Set summary bit indicates that an enabled operation event has occurred.

Serial polling and SRQ

Any enabled event summary bit that goes from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ has occurred and what caused it. If an SRQ occurs, the program can, for example, branch to an appropriate subroutine that will service the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit stays set until all Status Byte Register summary bits are reset.

SPE, SPD (serial polling)

For the GPIB interface only, the SPE and SPD general bus commands are used to serial poll the System Switch/Multimeter. Serial polling obtains the serial poll byte (status byte). Typically, serial polling is used by the controller to determine which of several instruments has requested service with the SRQ line.

Service requests

Service requests (SRQs) affect both the GPIB and the VXI-11 connections. On a GPIB connection, the SRQ line is asserted. On a VXI-11 connection, an SRQ event is generated.

Status byte and service request commands

The commands to program and read the Status Byte Register and Service Request Enable Register are listed in Status byte and service request commands (on page D-21). Note that the table includes both common commands and their script command equivalents. For details on programming and reading registers, see Programming enable and transition registers (on page D-15) and Reading registers (on page D-16).

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`).

Status Byte and Service Request Enable Register commands

Command	Description
`*STB?` or `print(status.condition)`	Read the Status Byte Register.
`*SRE <mask>` or `status.request_enable = <mask>`	Program the Service Request Enable Register where `<mask>` = 0 to 255.
`*SRE?` or `print(status.request_enable)`	Read the Service Request Enable Register.

Enable and transition registers

In general, there are three types of user-writable registers that are used to configure which bits feed the register summary bit and when it occurs. The registers are identified in each applicable command (as listed in TSP commands (on page 8-10)) as follows:

- **Enable register** (identified as `.enable` in each attribute's command listing): Allows various associated events to be included in the summary bit for the register.
- **Negative-transition register** (identified as `.ntr` in each attributes command listing): A particular bit in the event register will be set when the corresponding bit in the NTR is set, and the corresponding bit in the condition register transitions from 1 to 0.
- **Positive-transition register** (identified as `.ptr` in each attributes command listing): A particular bit in the event register will be set when the corresponding bit in the PTR is set, and the corresponding bit in the condition register transitions from 0 to 1.

Controlling node and SRQ enable registers

Attributes to control system node and service request (SRQ) enable bits and read associated registers are summarized in the Status byte and service request enable registers (see "Status Byte Register overview" on page D-4). For example, either of the following will set the system node QSB enable bit:

```
status.node_enable = status.QSB
status.node_enable = 8
```

TSP-Link system status

NOTE

TSP-Link® is not available on the Models 2604A/2614A/2634A.

The TSP-Link® expansion interface allows instruments to communicate with each other. The test system can be expanded to include up to 32 TSP-enabled instruments. In a TSP-Link system, one node (instrument) is the master and the other nodes are the subordinates. The master can control the other nodes (subordinates) in the system. See TSP-Link system expansion interface (on page 7-45) for details about the TSP-Link system.

The system summary registers, shown in the System summary and standard event registers (see "System summary bit (System register)" on page D-5), are shared by all nodes in the TSP-Link system. A status event that occurs at a subordinate node can generate an SRQ (service request) in the master node. After detecting the service request, your program can then branch to an appropriate subroutine that will service the request. See Status byte and service request (SRQ) (on page D-17) for details.

Status model configuration example

The following example illustrates the status model configuration for a TSP-Link system. In this example, a Node 15 thermal aspect event will set the RQS bit of the Status Byte of the master Node.

When the interlock event occurs on Node 15, the following sequence of events will occur:

1. On Node 15, with Bit B1 of the Questionable event register enabled, when the interlock event occurs, Bit B1 bit sets (`status.questionable.condition`) which causes Bit B1 to be set in `status.questionable.event`. This in turn causes the Questionable event summary bit (QSB) to set.
2. With QSB set, and Bit B3 of the System node enabled (`status.node_enable`), Bit B3 of the Status Byte register (Node 15) sets. This in turn causes the System node summary bit to set.
3. With the System node summary bit set, and Bit B1 of the System2 summary event register enabled (which is Node 15), Bit B1 of the System2 register sets. This in turn causes the System2 event summary bit (EXT) to set.
4. With EXT set, and Bit B0 of the System summary event register enabled, Bit B0 of the System register sets. This in turn causes the System event summary bit (SSB) to set.
5. With SSB set, and Bit B1 of the Service request enable register enabled, Bit B6 of the Status Byte register sets. This in turn initiates a request for service (SRQ).
6. When your program performs the next serial poll of the Master Node, it will detect the interlock event and can branch to a routine to service the request.

NOTE

The System Summary Registers are shared by all nodes in the TSP-Link system. When a bit in a system register of Node 15 sets, the same bit in the master node system register also sets.

The following commands (sent from the master node) enable the appropriate register bits for the above example:

Node 15 status registers: The following commands enable the events for Node 15:

```
node[15].status.questionable.enable = status.questionable.S1INL
node[15].status.node_enable = status.QSB
```

The affected status registers for the above commands are indicated by labels (1) and (2) (see the "TSP-Link status model configuration example" figure below).

System registers: The following commands enable the required system summary bits for Node 15:

```
status.system2.enable = status.system2.NODE15
status.system.enable = status.system.EXT
```

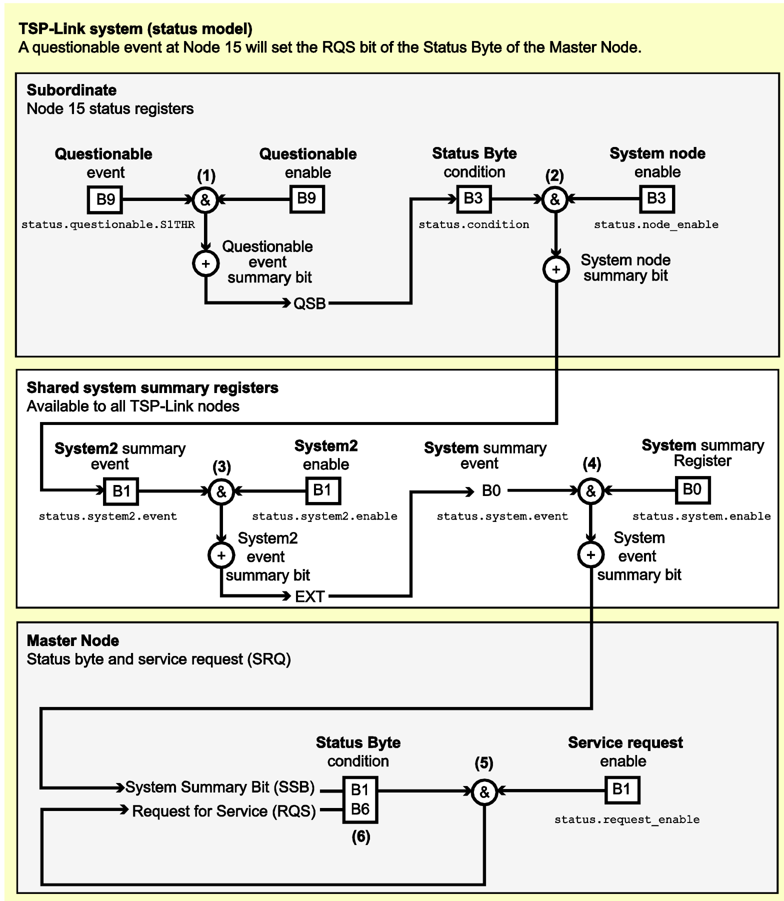
The affected system registers for the above commands are indicated by labels (3) and (4) (see the "TSP-Link status model configuration example" figure below).

Master Node service request: The following command enables the service request for the measurement event:

```
status.request_enable = status.SSB
```

The affected status register for the above command is indicated by labels (5) and (6) (see the "TSP-Link status model configuration example" figure below).

Figure 169: TSP-Link status model configuration example



Index

D

E

W