

Models 707B and 708B Switching Matrix

Reference Manual

707B-901-01 Rev. A / August 2010



Models 707B and 708B Switching Matrix Reference Manual

© 2010, Keithley Instruments, Inc.

Cleveland, Ohio, U.S.A.

All rights reserved.

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

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

The Lua 5.0 software and associated documentation files are copyright © 1994-2008, Tecgraf, PUC-Rio. Terms of license for the Lua software and associated documentation can be accessed at the [Lua licensing site](http://www.lua.org/license.html) (<http://www.lua.org/license.html>).

Document number: 707B-901-01 Rev. A / August 2010

Safety Precautions



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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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


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


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


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

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

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


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

The  symbol on an instrument means caution, risk of danger. The user should 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 danger. Use standard safety precautions to avoid personal contact with these voltages.

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

Table of Contents

Introduction	1-1
Contact information	1-1
Overview	1-1
Warranty information.....	1-1
General operation	2-1
Rear panel overview	2-1
Model 707B Rear panel	2-2
Model 708B Rear panel	2-2
Wiring	2-2
Communication connections	2-3
Digital I/O port	2-7
Power-up	2-10
Line power connection	2-10
Power-up sequence	2-11
Front-panel operation.....	2-11
Model 707B front panel	2-12
Model 708B front panel	2-12
Display	2-13
Crosspoint display (Model 707B only)	2-15
Selecting channels from the front panel	2-17
Keys and navigation wheel.....	2-18
Menu options.....	2-21
Communication interfaces.....	2-24
USB.....	2-24
GPIB	2-31
LAN communications	2-36
Supplied software.....	2-58
Keithley I/O layer.....	2-59
Addressing instruments with VISA	2-64
Using the web interface.....	2-69
Introduction	2-69
Card pages.....	2-71
Scan Builder page.....	2-76
TSB Embedded.....	2-81
Admin page.....	2-82
Unit page.....	2-83
LXI page.....	2-84
Switch operation.....	2-87
Working with channels	2-87
Reset a channel	2-105
Pseudocards	2-106
Save the present configuration.....	2-106

Functions and features	3-1
Scanning and triggering	3-1
Trigger model	3-1
Trigger model components.....	3-3
Trigger model events and associated commands	3-4
Scan and step counts.....	3-4
Basic scan procedure.....	3-5
Changing attributes of an existing scan	3-6
Front-panel scanning	3-6
Foreground and background scan execution	3-7
Include multiple channels in a single scan step.....	3-7
Remote interface scanning	3-8
Scan and trigger commands	3-8
Scanning examples.....	3-9
Hardware trigger modes.....	3-12
Falling edge trigger mode	3-14
Rising edge master trigger mode	3-15
Rising edge acceptor trigger mode	3-15
Either edge trigger mode.....	3-16
Understanding synchronous triggering modes	3-17
Synchronous master trigger mode	3-18
Synchronous acceptor trigger mode	3-19
Synchronous trigger mode	3-20
Events	3-21
Event blenders	3-22
Theory of operations	4-1
Models 707B and 708B theory of operations overview	4-1
Mainframe	4-1
Important test system safety information.....	4-2
Instrument fan (Model 707B only)	4-3
AC power entry	4-3
Universal power supply	4-3
Microprocessor board	4-3
Communication interfaces.....	4-4
Trigger and control interfaces.....	4-4
Backplane	4-4
Front panel	4-6
Upper crosspoint display (Model 707B only)	4-6
Remote commands.....	5-1
Introduction to remote operation	5-1
Controlling the instrument by sending individual command messages	5-1
Queries	5-3
Data retrieval commands	5-3
Information on scripting and programming.....	5-3
About ICL commands.....	5-4

beeper functions and attributes	5-4
bit functions	5-4
channel functions and attributes.....	5-5
createconfigscript function.....	5-8
dataqueue functions and attributes	5-9
delay function	5-9
digio functions and attributes.....	5-9
display functions and attributes	5-10
errorqueue functions and attributes.....	5-10
eventlog functions and attributes.....	5-11
exit function	5-11
format attributes	5-11
gpib attribute	5-11
lan functions and attributes	5-12
localnode functions and attributes.....	5-13
make accessor functions.....	5-13
memory functions.....	5-13
opc function.....	5-13
print functions.....	5-13
reset function.....	5-13
scan functions and attributes.....	5-14
script functions and attributes.....	5-14
slot[X] attributes	5-15
status functions and attributes.....	5-15
timer functions.....	5-15
trigger functions and attributes	5-16
tslink functions and attributes.....	5-16
tspnet functions and attributes	5-17
userstring functions	5-17
waitcomplete function.....	5-17
Overview of instrument drivers	5-18
Instrument driver types.....	5-18
VXIbP drivers.....	5-19
Interchangeable Virtual Instruments (IVI) style drivers.....	5-19
LabVIEW drivers	5-20
Obtaining instrument drivers	5-20
Instrument driver examples.....	5-20
Migrating from Models 707A and 708A.....	5-21
Migrating Model 707A or 708A programs to Model 707B or 708B	5-21
DDC to ICL command equivalencies	5-22

Instrument programming

Fundamentals of scripting for TSP.....	6-1
What is a script?.....	6-2
Runtime and nonvolatile memory storage of scripts.....	6-2
What can be included in scripts?.....	6-3
Commands that cannot be used in scripts	6-3
Manage scripts.....	6-3
Working with scripts in nonvolatile memory.....	6-9
Programming examples	6-13
Fundamentals of programming for TSP.....	6-15
Introduction	6-15
What is Lua?	6-15
Lua basics	6-15
Standard libraries.....	6-30
Programming example	6-34

Using Test Script Builder (TSB)	6-35
Installing the TSB software.....	6-35
Project Navigator.....	6-35
Script Editor.....	6-36
Programming interaction	6-36
Advanced scripting for TSP	6-36
Global variables and the script.user.scripts table	6-36
Create a script using the script.new command.....	6-38
Restore a script to the runtime environment.....	6-41
Rename a script.....	6-41
Delete user scripts from the instrument.....	6-43
Memory considerations for the runtime environment	6-44
TSP-Link system and running parallel test scripts	6-45
TSP-Link system	6-45
TSP-Link nodes.....	6-46
Connect the TSP-Link cable.....	6-46
Initialization	6-47
Introduction to TSP advanced features	6-49
Using groups to manage nodes on TSP-Link network	6-50
Running parallel test scripts	6-52
Using the data queue for real-time communication	6-53
Copying test scripts across the TSP-Link network	6-54
Removing stale values from the reading buffer	6-54
TSP-Net	6-55
Overview	6-55
TSP-Net Capabilities.....	6-55
Using TSP-Net with any Ethernet-enabled device	6-56
Using TSP-Net vs. TSP-Link for communication with TSP-enabled devices	6-58
Instrument Control Library (ICL) - General device control.....	6-59
Instrument Control Library — TSP-Enabled device control.....	6-59
Example: Using tspnet commands.....	6-60

Command reference 7-1

Command programming notes	7-1
Placeholder characters	7-1
Syntax rules	7-1
Logical instruments	7-3
Time and date values.....	7-3
Using the Instrument Control Library documentation	7-4
Command name and standard parameters summary	7-4
Command usage.....	7-6
Command details	7-7
Example section.....	7-7
Also see: Related commands and information	7-8
Instrument Control Library (ICL) command reference	7-8
beeper.beep().....	7-8
beeper.enable	7-9
bit.bitand()	7-9
bit.bitor()	7-10
bit.bitxor()	7-11
bit.clear()	7-11
bit.get()	7-12
bit.getfield()	7-13
bit.set()	7-14
bit.setfield().....	7-15

bit.test()	7-16
bit.toggle()	7-17
channel.clearforbidden()	7-18
channel.close()	7-19
channel.connectrule	7-20
channel.connectsequential	7-21
channel.createspecifier()	7-23
channel.exclusiveclose()	7-24
channel.exclusiveslotclose()	7-25
channel.getclose()	7-27
channel.getcount()	7-28
channel.getdelay()	7-29
channel.getforbidden()	7-30
channel.getlabel()	7-31
channel.getlabelcolumn()	7-32
channel.getlabelrow()	7-33
channel.getstate()	7-34
channel.gettype()	7-35
channel.open()	7-35
channel.pattern.catalog()	7-37
channel.pattern.delete()	7-38
channel.pattern.getimage()	7-38
channel.pattern.setimage()	7-39
channel.pattern.snapshot()	7-41
channel.reset()	7-43
channel.setdelay()	7-44
channel.setforbidden()	7-45
channel.setlabel()	7-46
channel.setlabelcolumn()	7-47
channel.setlabelrow()	7-49
createconfigscript()	7-50
dataqueue.add()	7-51
dataqueue.CAPACITY	7-52
dataqueue.clear()	7-52
dataqueue.count	7-53
dataqueue.next()	7-54
delay()	7-55
digio.readbit()	7-55
digio.readport()	7-56
digio.trigger[N].assert()	7-57
digio.trigger[N].clear()	7-57
digio.trigger[N].EVENT_ID	7-58
digio.trigger[N].mode	7-58
digio.trigger[N].overrun	7-60
digio.trigger[N].pulsewidth	7-60
digio.trigger[N].release()	7-61
digio.trigger[N].reset()	7-62
digio.trigger[N].stimulus	7-62
digio.trigger[N].wait()	7-64
digio.writebit()	7-64
digio.writeport()	7-65
digio.writeprotect	7-66
display.clear()	7-67
display.getannunciators()	7-67
display.getcursor()	7-68
display.getlastkey()	7-69
display.gettext()	7-70
display.inputvalue()	7-73
display.loadmenu.add()	7-74
display.loadmenu.catalog()	7-76
display.loadmenu.delete()	7-76

display.locallockout	7-77
display.menu().....	7-77
display.prompt()	7-78
display.screen	7-80
display.sendkey()	7-80
display.setcursor().....	7-81
display.settext().....	7-82
display.trigger.EVENT_ID	7-84
display.waitkey().....	7-84
errorqueue.clear()	7-85
errorqueue.count.....	7-86
errorqueue.next()	7-86
eventlog.all().....	7-87
eventlog.clear()	7-88
eventlog.count.....	7-89
eventlog.enable.....	7-90
eventlog.next()	7-90
eventlog.overwritemethod	7-91
exit()	7-92
format.asciiprecision	7-93
format.byteorder	7-93
format.data	7-94
gettimezone()	7-96
gpib.address.....	7-97
lan.applysettings()	7-98
lan.config.dns.address[N].....	7-98
lan.config.dns.domain	7-99
lan.config.dns.dynamic.....	7-100
lan.config.dns.hostname	7-101
lan.config.dns.verify	7-101
lan.config.gateway	7-102
lan.config.ipaddress	7-104
lan.config.method.....	7-104
lan.config.subnetmask	7-105
lan.lxidomain	7-106
lan.nagle	7-106
lan.reset()	7-107
lan.restoredefaults()	7-107
lan.status.dns.address[N].....	7-108
lan.status.dns.name	7-109
lan.status.duplex	7-109
lan.status.gateway	7-110
lan.status.ipaddress	7-110
lan.status.macaddress	7-111
lan.status.port.dst.....	7-111
lan.status.port.rawsocket	7-112
lan.status.port.telnet.....	7-112
lan.status.port.vxi11	7-113
lan.status.speed	7-113
lan.status.subnetmask	7-114
lan.trigger[N].assert()	7-114
lan.trigger[N].clear()	7-115
lan.trigger[N].connect().....	7-116
lan.trigger[N].connected	7-117
lan.trigger[N].disconnect()	7-118
lan.trigger[N].EVENT_ID	7-118
lan.trigger[N].ipaddress	7-119
lan.trigger[N].mode.....	7-119
lan.trigger[N].overrun	7-121
lan.trigger[N].protocol.....	7-122
lan.trigger[N].pseudostate	7-122

lan.trigger[N].stimulus	7-123
lan.trigger[N].wait()	7-125
localnode.define.*	7-125
localnode.description	7-126
localnode.execute()	7-127
localnode.getglobal()	7-128
localnode.model	7-128
localnode.password	7-129
localnode.prompts	7-129
localnode.prompts4882	7-130
localnode.reset()	7-131
localnode.revision	7-131
localnode.serialno	7-132
localnode.setglobal()	7-133
localnode.showerrors	7-133
makegetter()	7-134
makesetter()	7-135
memory.available()	7-135
memory.used()	7-137
opc()	7-137
print()	7-138
printbuffer()	7-138
printnumber()	7-140
reset()	7-140
scan.abort()	7-141
scan.add()	7-142
scan.addimagestep()	7-143
scan.background()	7-144
scan.bypass	7-145
scan.create()	7-146
scan.execute()	7-147
scan.list()	7-148
scan.mode	7-150
scan.reset()	7-150
scan.scancount	7-151
scan.state()	7-152
scan.stepcount	7-153
scan.trigger.arm.clear()	7-153
scan.trigger.arm.set()	7-154
scan.trigger.arm.stimulus	7-154
scan.trigger.channel.clear()	7-156
scan.trigger.channel.set()	7-156
scan.trigger.channel.stimulus	7-157
scan.trigger.clear()	7-158
script.anonymous	7-159
script.delete()	7-159
script.new()	7-160
script.newautorun()	7-161
script.restore()	7-162
script.run()	7-162
script.user.catalog()	7-163
scriptVar.autorun	7-163
scriptVar.list()	7-164
scriptVar.name	7-164
scriptVar.run()	7-166
scriptVar.save()	7-166
scriptVar.source	7-167
settime()	7-167
settimezone()	7-168
slot[X].idn	7-169
slot[X].poles.four	7-170

slot[X].poles.one	7-171
slot[X].poles.two	7-171
slot[X].pseudocard	7-172
status.condition	7-173
status.node_enable	7-175
status.node_event	7-177
status.operation.*	7-178
status.operation.user.*	7-180
status.questionable.*	7-182
status.request_enable	7-184
status.request_event	7-186
status.reset()	7-187
status.standard.*	7-188
status.system.*	7-190
status.system2.*	7-192
status.system3.*	7-194
status.system4.*	7-196
status.system5.*	7-198
timer.measure.t()	7-200
timer.reset()	7-201
trigger.blender[N].clear()	7-201
trigger.blender[N].EVENT_ID	7-202
trigger.blender[N].orenable	7-202
trigger.blender[N].overrun	7-203
trigger.blender[N].reset()	7-203
trigger.blender[N].stimulus[M]	7-204
trigger.blender[N].wait()	7-205
trigger.clear()	7-206
trigger.EVENT_ID	7-206
trigger.timer[N].clear()	7-207
trigger.timer[N].count	7-207
trigger.timer[N].delay	7-208
trigger.timer[N].delaylist	7-209
trigger.timer[N].EVENT_ID	7-209
trigger.timer[N].overrun	7-210
trigger.timer[N].passthrough	7-211
trigger.timer[N].reset()	7-211
trigger.timer[N].stimulus	7-212
trigger.timer[N].wait()	7-213
trigger.wait()	7-214
tslink.group	7-214
tslink.master	7-215
tslink.node	7-215
tslink.readbit()	7-216
tslink.readport()	7-216
tslink.reset()	7-217
tslink.state	7-218
tslink.trigger[N].assert()	7-218
tslink.trigger[N].clear()	7-219
tslink.trigger[N].EVENT_ID	7-219
tslink.trigger[N].mode	7-220
tslink.trigger[N].overrun	7-221
tslink.trigger[N].pulsewidth	7-222
tslink.trigger[N].release()	7-223
tslink.trigger[N].reset()	7-223
tslink.trigger[N].stimulus	7-224
tslink.trigger[N].wait()	7-225
tslink.writebit()	7-226
tslink.writeport()	7-226
tslink.writeprotect	7-227
tspnet.clear()	7-228

tspnet.connect()	7-229
tspnet.disconnect()	7-230
tspnet.execute()	7-230
tspnet.idn()	7-232
tspnet.read()	7-232
tspnet.readavailable()	7-233
tspnet.reset()	7-234
tspnet.termination()	7-235
tspnet.timeout	7-236
tspnet.tsp.abort()	7-236
tspnet.tsp.abortonconnect	7-237
tspnet.tsp.rhtablecopy()	7-238
tspnet.tsp.runscript()	7-239
tspnet.write()	7-240
userstring.add()	7-240
userstring.catalog()	7-241
userstring.delete()	7-242
userstring.get()	7-242
waitcomplete()	7-243

Troubleshooting guide 8-1

Troubleshooting guide.....	8-1
Error and status messages	8-1
Error summary	8-1
Effects of errors on scripts	8-2
Error queue remote commands.....	8-2
USB troubleshooting	8-2
Check driver for the USB Test and Measurement Device.....	8-2
Troubleshooting GPIB interfaces	8-5
Timeout errors.....	8-5
Troubleshooting LAN interfaces.....	8-5
Verify connections and settings.....	8-6
Use Ping to test the connection.....	8-6
Open ports on firewalls	8-7
Web page problems	8-7
LXI LAN status indicator.....	8-8
Initialize the LAN configuration.....	8-8
Install LXI Discovery Browser software on your computer	8-8
Communicate using VISA communicator	8-9
WireShark	8-9
Testing the display, keys, and channel matrix	8-9
Verify front panel key operation.....	8-9
Verify display operation.....	8-10
Verify crosspoint display operation (707B only)	8-10
Update drivers	8-10
Contacting support.....	8-11

Frequently asked questions 9-1

How do I get my LAN or web connection to work?	9-1
Why can't I close a channel?	9-1

How do I know if an error has occurred on my instrument? 9-2

How do I find the serial number and firmware version of the instrument?..... 9-3

Next steps..... 10-1

 Additional Models 707B and 708B information 10-1

MaintenanceA-1

 Upgrading firmware..... A-1

 Check fan status A-2

 Fuse replacement A-2

Using Models 707A and 708A compatibility modeB-1

 Model A to Model B differences B-1

 Front-panel relay closure indicators B-1

 Timing issues B-2

 Digital interface B-2

 Memory setups..... B-2

 Models 707A and 708A commands B-3

Status modelC-1

 Overview C-1

 Status Byte register C-2

 Status model diagrams C-3

 Status Byte register overview..... C-4

 Measurement summary bit (Measurement event register) C-5

 System summary bit (System register) C-5

 Error available bit (Error or Event queue)..... C-7

 Questionable summary bit (Questionable event register) C-7

 Message available bit (Output queue)..... C-8

 Event summary bit (ESB register) C-9

 Master summary status bit (MSS bit register) C-10

 Operation summary bit (Operation event register) C-11

 Status function summary..... C-12

 Clearing registers and queues C-13

 Startup state C-13

 Programming and reading registers..... C-13

 Programming enable and transition registers..... C-13

 Reading registers C-14

 Register programming example C-15

 Status byte and service request (SRQ) C-15

 Service request enable register..... C-15

 Status byte register C-16

 Serial polling and SRQ C-17

 SPE, SPD (serial polling) C-18

 Service requests (SRQs) and connections C-18

Status byte and service request commands.....C-18
Enable and transition registers.....C-19
Controlling node and SRQ enable registers.....C-19
TSP-Link system status C-19
Status model configuration exampleC-19

Index.....Index-1

Limitation of Warranty

Introduction

In this section:

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

Contact information

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) (<http://www.keithley.com>).

Overview

The Model 707B or 708B provides outstanding low-current matrix capability and lets you control up to 576 matrix crosspoints in real time. Their large matrix format makes them well suited for your large ATE system applications, such as semiconductor device characterization, wafer level reliability, parallel test, and modeling.

The Model 707B can host up to six test cards. The Model 708B is a single slot chassis.

Warranty information

Detailed warranty information is located at the back of this manual. Should your Model 707B or 708B require warranty service, contact the Keithley Instruments representative or authorized repair facility in your area for further information. When returning the instrument for repair, be sure to complete the service form at the back of this manual and give it to the repair facility with all relevant information.

NOTE

The service form requires the serial number of the Model 707B or 708B. The serial number is located on the rear panel of the instrument.

**WARNING**

Before removing or installing switching cards, make sure you turn off the Model 707B or 708B and disconnect the line cord. Also, remove any other external power connected to the instrument or switching cards.

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

General operation

In this section:

Rear panel overview	2-1
Wiring	2-2
Power-up	2-10
Front-panel operation	2-11
Communication interfaces	2-24
Using the web interface	2-69
Switch operation	2-87

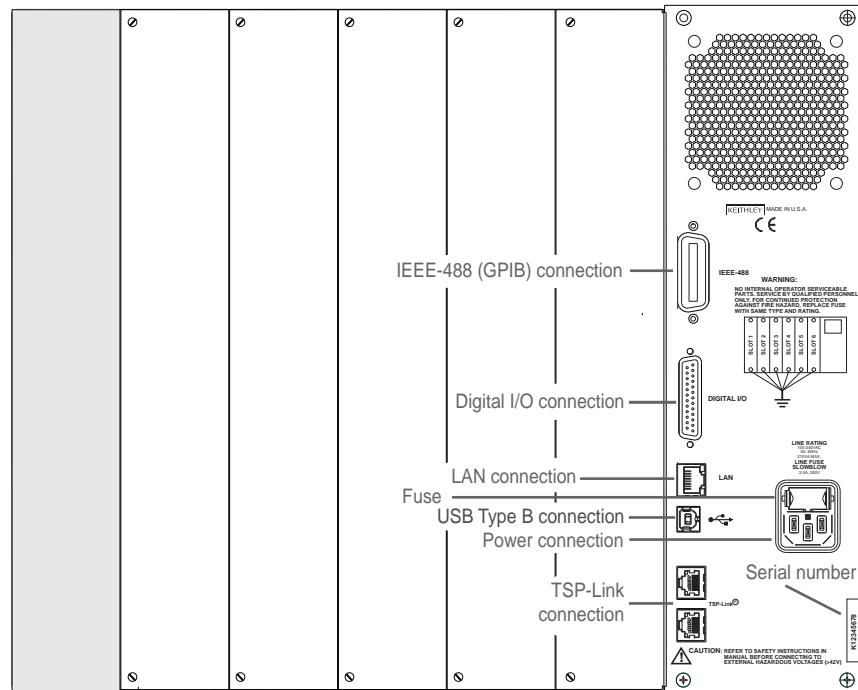
Rear panel overview

You make power and communications connections to the rear panel of the instrument. The connections available are described in the following table. The following figures show the locations of the connections.

Rear panel options	
Option	Description
Slots	Use the slots in the Keithley Instruments Models 707B and 708B for the switching cards. The Model 707B can accept up to six switching cards. The Model 708B can accept one switching card. If a slot does not contain a card, make sure to cover the slot with a slot cover. For model and firmware version information on the installed cards, press the SLOT key.
IEEE-488	IEEE-488 (GPIB) connector. See GPIB quick start (on page 2-31).
Digital I/O	Digital input/output connector. See Digital I/O port (on page 2-7) for connection information.
LAN	Ethernet (LAN) connector. See Connect the LAN cable (on page 2-4).
Fuse	Line fuse. Model 707B fuse rating is Slow Blow 2.0A, 250V. Model 708B is Slow Blow 1.0 A 250V. To replace the fuse, see Fuse replacement (on page A-2).
USB (Type B)	USB communication interface connection. See Connect the USB cable (on page 2-3).
Power	Using the supplied line cord, connect to a grounded AC power outlet. See Line power connection (on page 2-10) for connection details.
TSP-LINK	Use with TSP-Link [®] cable to expand the system. See Connect the TSP-Link cable (on page 2-6).
Serial number	Serial number of the instrument.

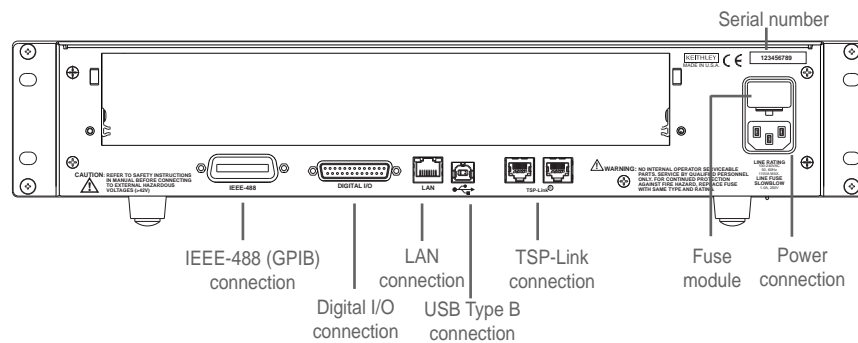
Model 707B Rear panel

Figure 1: Model 707B rear panel



Model 708B Rear panel

Figure 2: Model 708B full rear panel



Wiring

This section describes communication, digital I/O, and power connections.

Note that all signal wiring to devices and instruments is done through the switch cards. Please refer to the switch card manuals for additional information.

Communication connections

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](#) (on page 2-24).

Connect the USB cable

To connect the USB cable:

Connect the Type B end of the USB cable to the connector on the back of the instrument (shown below).

Figure 3: Model 707B rear panel USB connection

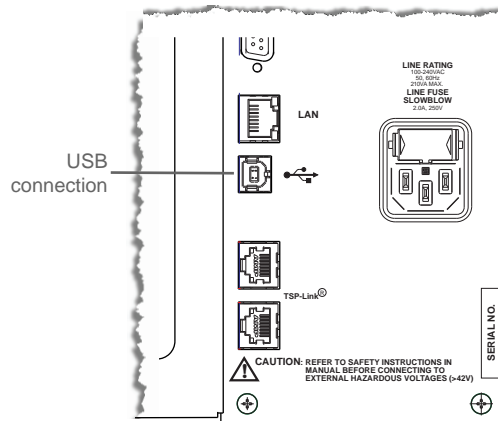
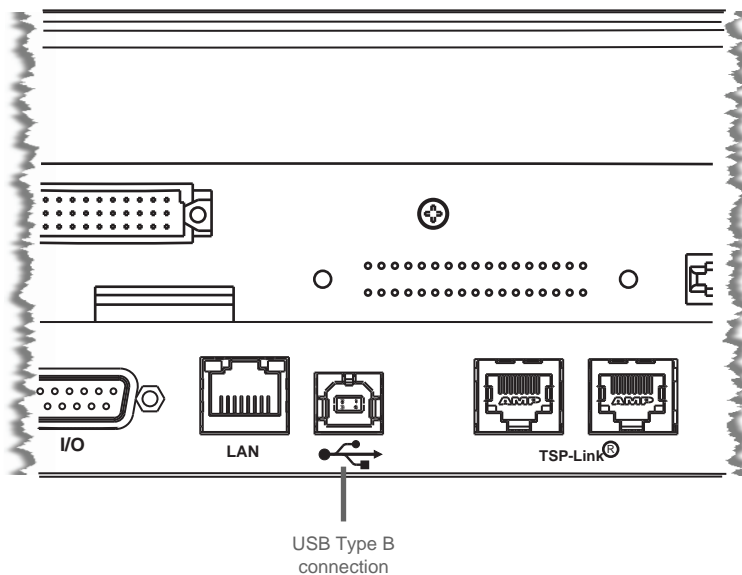


Figure 4: Model 708B rear panel USB connection



Connect the LAN cable

Connect the LAN 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.

The location of the LAN connector on the instrument is shown below.

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.

Figure 5: Model 707B rear panel LAN connection

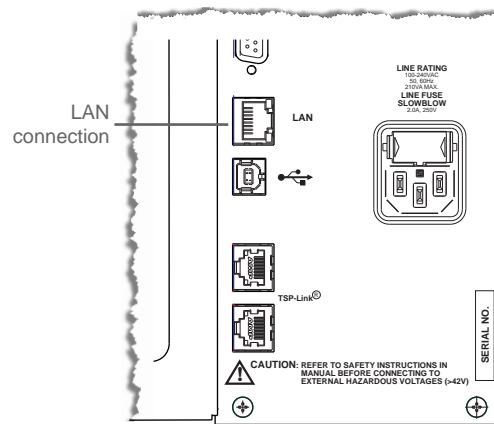
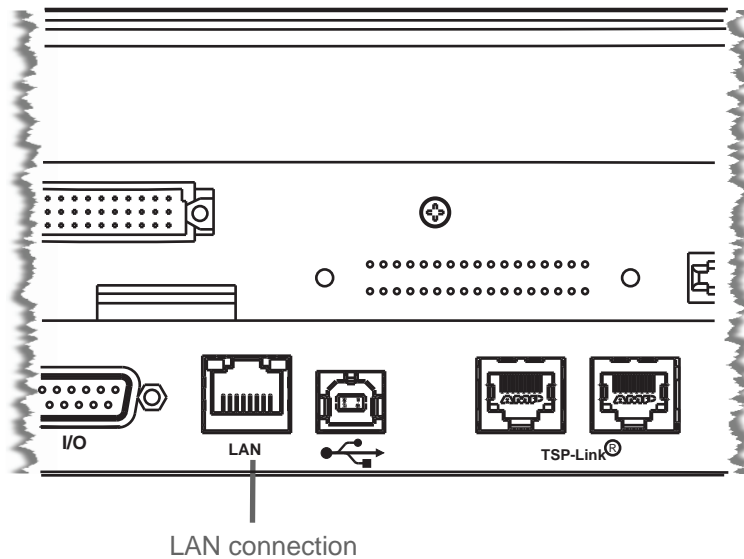


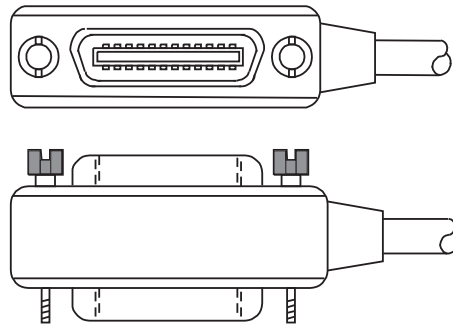
Figure 6: Model 708B rear panel LAN connection



Connect the GPIB cable

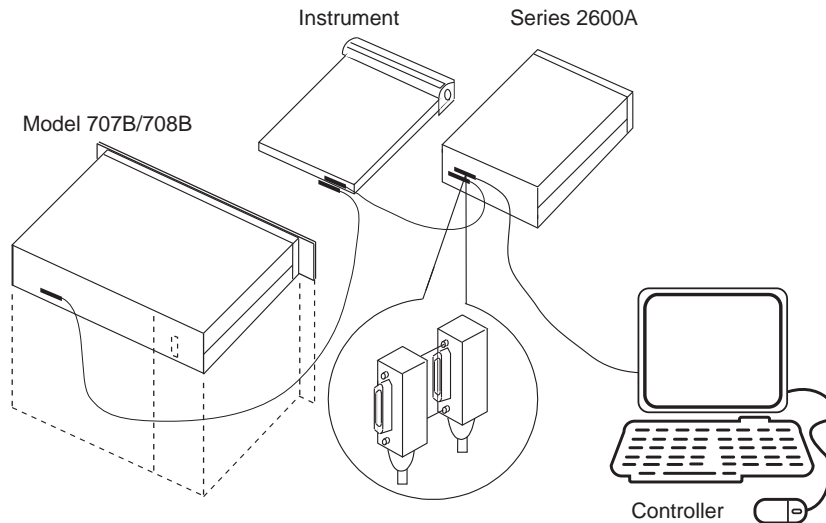
To connect a Model 707B or 708B to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown below.

Figure 7: IEEE-488 connector



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

Figure 8: IEEE-488 connections

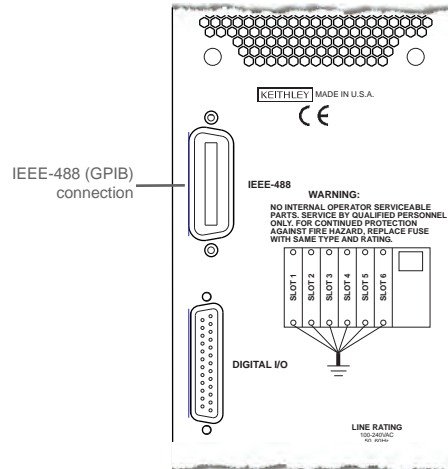


⚠ CAUTION

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 Model 707B or 708B 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 9: Model 707B rear panel IEEE-488 connection



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.

Connect the TSP-Link cable

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.

NOTE

For an example of setting up a TSP-Linked system, see "Working with a Series 2600A" in the Models 707B and 708B User's Manual.

Figure 10: Model 708B rear panel TSP-Link connection

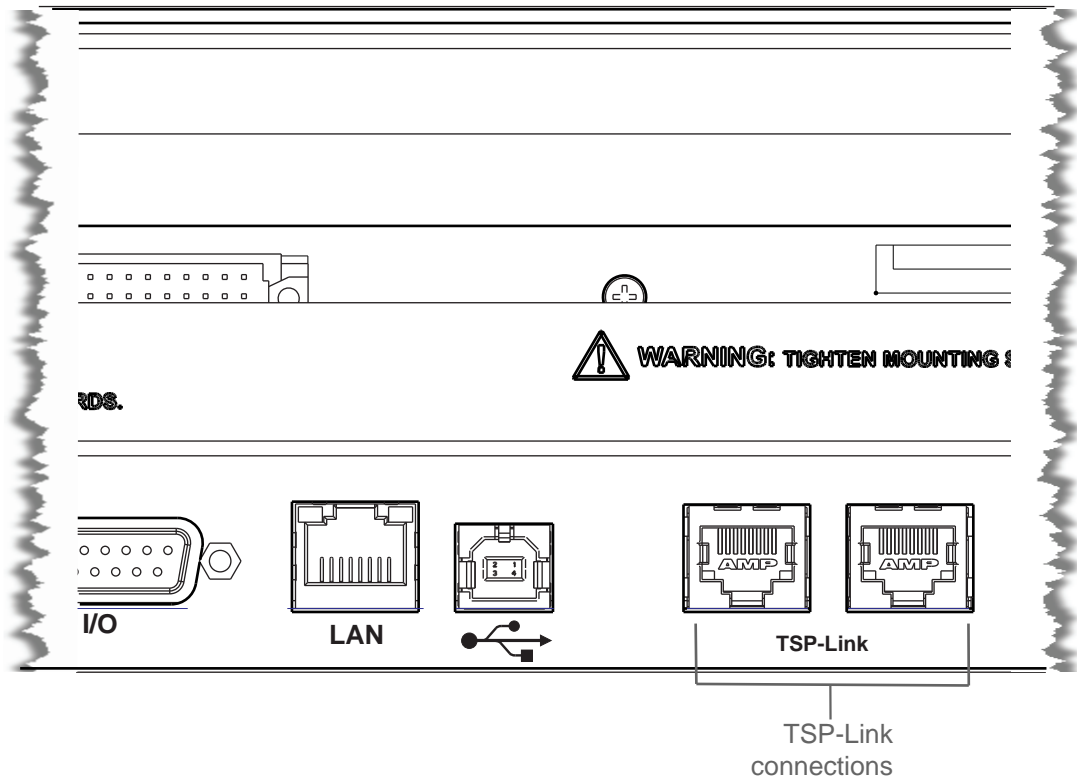
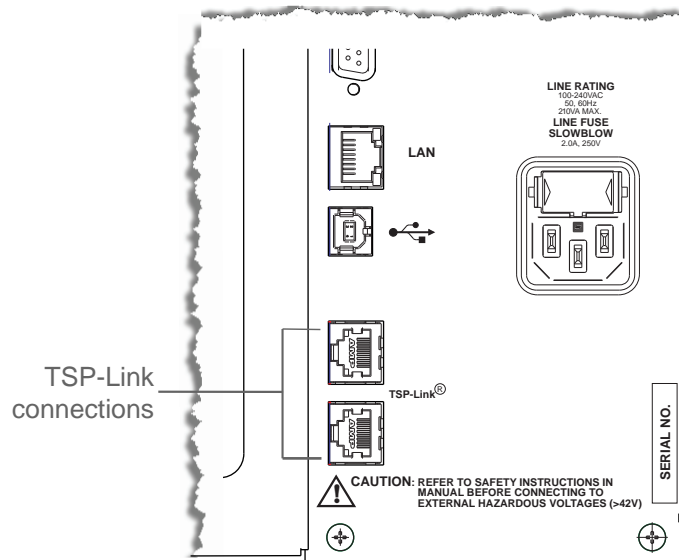
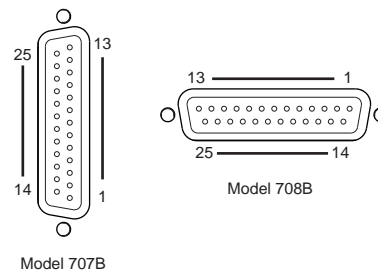


Figure 11: Model 707B rear panel TSP-Link connection



Digital I/O port

The Models 707B and 708B 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 12: Models 707B and 708B digital I/O ports

Pin	Description
1	Digital I/O #1
...	...
14	Digital I/O #14
15-21	Ground
22	+5V D (fused, 600 mA)
23	Not connected; pin reserved for future use
24	Not connected; pin reserved for future use
25	+5V D (fused, 600 mA)

NOTE

If you were using a Model 707A or 708A, see [Using Models 707A and 708A compatibility mode](#) (on page B-1).

Connecting cables

Use a cable equipped with a standard male DB-25 connector (Keithley Instruments part number CA-126-1).

Digital I/O lines (pins 1 through 14)

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

+5 V output

The digital I/O port provides a +5 V output that is used to drive external logic circuitry. Maximum current output for this line is 600 mA. This line is protected by a self-resetting fuse (one hour recovery time).

Controlling digital I/O lines

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.

You can use digital I/O lines for both input and output.

NOTE

You must write a 1 to all digital I/O lines that are to be used as inputs.

NOTE

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 3-1) for information.

NOTE

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**. Press the **ENTER** key or press the navigation wheel.
2. Select **DIGIO-OUTPUT**. Press the **ENTER** key or the navigation wheel.
3. Set the decimal value as required to set digital I/O lines. The range is 0 to 16,383 (see [Digital I/O bit weighting](#) (on page 2-9)). Press the **ENTER** key or the navigation wheel.
4. Press the **EXIT** key as needed to return to the main menu.

To write protect specific digital I/O lines from the front panel:

1. Press the **MENU** key, then select **DIGIO**. Press the **ENTER** key or the navigation wheel.
2. Select **WRITE-PROTECT**. Press the **ENTER** key or the navigation wheel.
3. Set the decimal value as required to write protect digital I/O lines. The range is 0 to 16,383 (see [Digital I/O bit weighting](#) (on page 2-9)). Press the **ENTER** key or the navigation wheel.
4. Press the **EXIT** key as needed to return to the main menu.
5. To remove write protection, repeat the above steps, but enter the original value in step 3.

To set digital I/O values from the remote communication interface:

Send the [digio.writeport\(\)](#) (on page 7-65) command.

To write-protect specific digital I/O lines from the remote communication interface:

Send the [digio.writeprotect](#) (on page 7-66) command.

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	1024	0x0400
12	B12	2048	0x0800
13	B13	4096	0x1000
14	B14	8192	0x2000

Power-up

Line power connection

Follow the procedure below to connect the Model 707B or 708B to line power and turn on the instrument.

The Model 707B or 708B operates from a line voltage of 100 V to 240 V at a frequency of 50 Hz or 60 Hz. Line voltage is automatically sensed (there are no switches to set). Make sure the operating voltage in your area is compatible.



WARNING

The power cord supplied with the Model 707B or 708B contains a separate ground wire for use with grounded outlets. When proper connections are made, instrument chassis is connected to power line ground through the ground wire in the power cord. Failure to use a grounded outlet may result in personal injury or death due to electric shock.



CAUTION

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

To connect the Model 707B or 708B to line power and turn on the instrument:

1. Make sure that the front panel power switch is in the off (O) position. See [Front-panel operation](#) (on page 2-11) for switch location.
2. Connect the female end of the supplied power cord to the power connection (AC receptacle) on the rear panel. See [Rear panel overview](#) (on page 2-1) for connector location.
3. Connect the other end of the power cord to a grounded AC outlet.
4. Turn on the instrument by pressing the front panel power switch to the on (I) position.

Power-up sequence

When the instrument is turned on, the instrument performs self-tests and momentarily lights all segments and indicators on the display. If a failure is detected, the instrument momentarily displays an error message. Error messages are listed in Error and status messages.


NOTE


If a problem develops while the instrument is under warranty, return it to Keithley Instruments, Inc., for repair. See the Warranty page at the back of this manual for more information.

If there are no errors, the following actions occur when the instrument is turned on, three dots are briefly displayed. On the Model 707B, the crosspoint display shows the text "Wait for Init to End." When initialization is complete, the bottom display shows "KEITHLEY Model 707B." The Model 708B displays "KEITHLEY Model 708B."

Front-panel operation

The front panel of the Keithley Instruments Model 707B or 708B contains the following items:

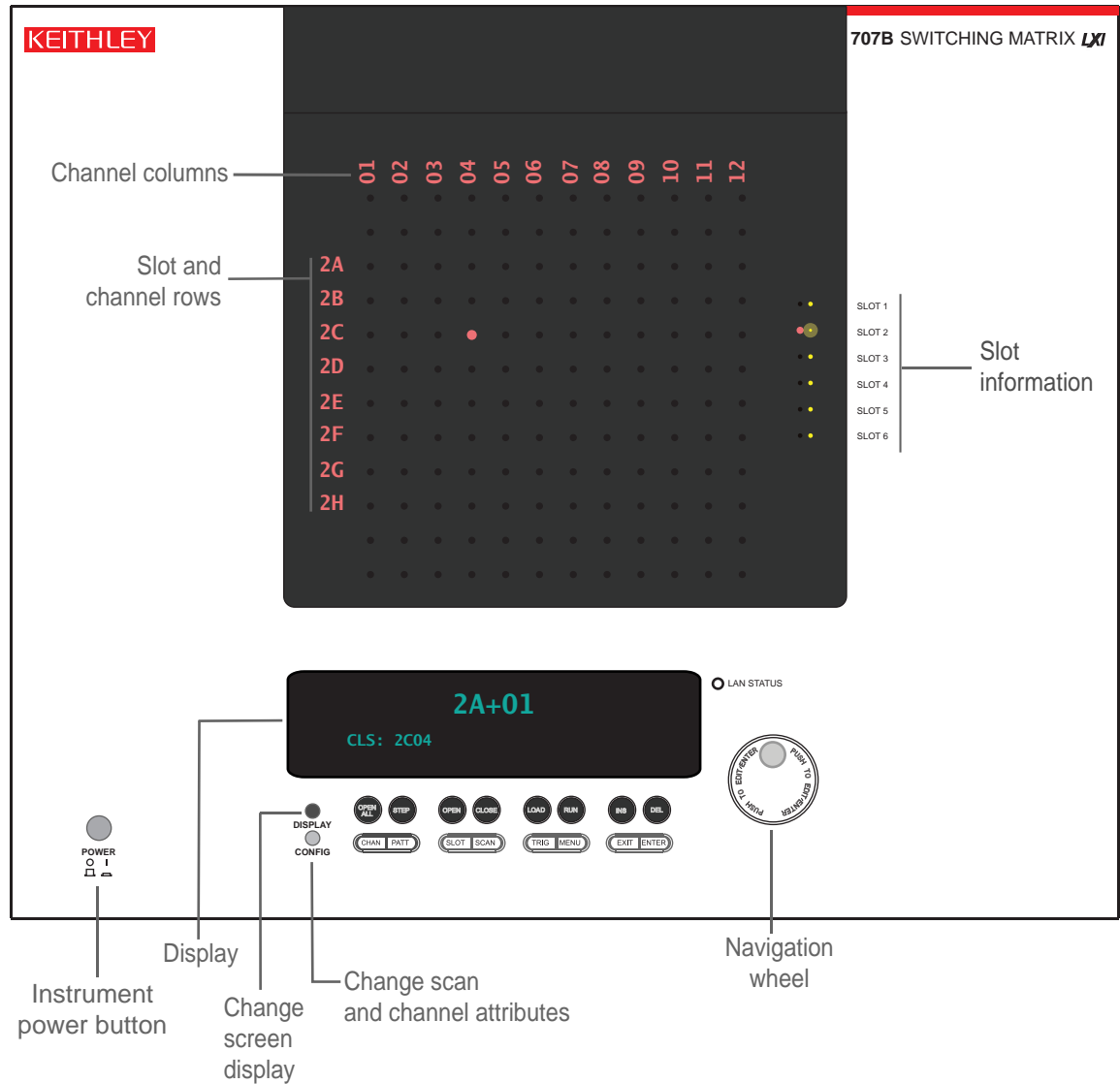
- The display
- The crosspoint display (Model 707B only)
- The keys and navigation wheel 
- The LAN status indicator
- The POWER button

You can use the keys, displays, and the navigation wheel  to change the selected channel or channel pattern. You can also use them to access, view, and edit the menu items. The crosspoint display on the Model 707B shows you which channels are opened and closed.

Model 707B front panel

The front panel of the Model 707B is shown below.

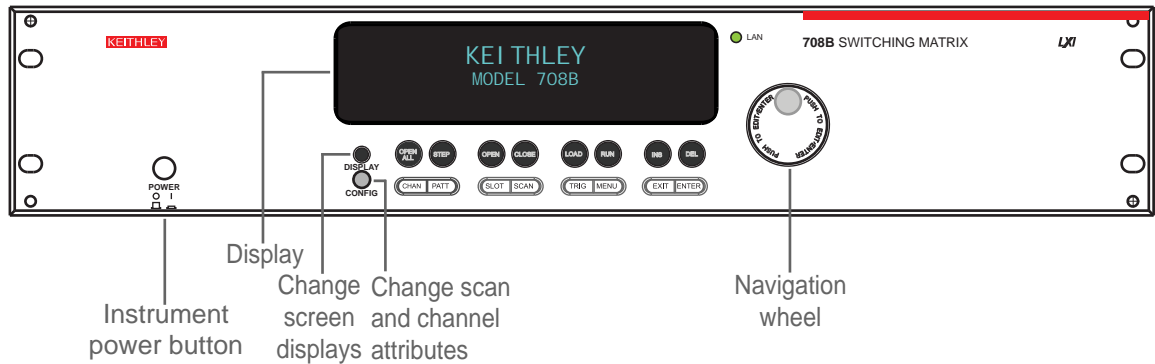
Figure 13: Model 707B front panel



Model 708B front panel

The front panel of the Model 708B is shown below.

Figure 14: Model 708B front panel



Display

NOTE

This section describes the front-panel display of the Model 708B and the bottom display of the Model 707B.

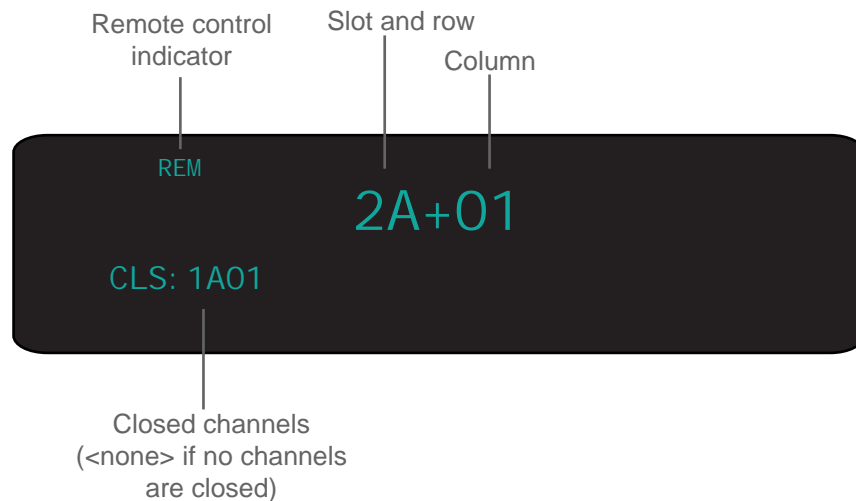
During operation, the display provides information about the selected channel, channel pattern, channel state, and errors. You can press **DISPLAY** to cycle between the display of the channel or pattern, the closed channel list, or a screen message.

During setup, the display shows menu choices that you can use to configure the instrument.

Display during operation

During operation, the display shows the control status (local or remote) and the current channel, and indicates if any channels are closed. An example is shown below.

If REM is not displayed, control is through the front panel.



The control status is shown in the upper left corner of the display. If REM is displayed, the instrument is being controlled remotely (through GPIB, LAN, or USB).

If you are connecting to the instrument through GPIB, you may also see the following indicators:

- TALK: Instrument is addressed to talk
- LSTN: Instrument is addressed to listen
- SRQ: Service request
- REM: Remote communication

By default, the top line of the display shows the slot, row, and column of the selected channel. If labels have been set up for your instrument, you might see four-character labels for your channels, such as GATE+SMU1. See [Set up labels](#) (on page 2-98) for information on setting up labels.

To change the selected channel:

1. Press the navigation wheel to select the row.
2. Turn the navigation wheel to go to a different row.
3. Press the navigation wheel again to select the column.
4. Turn the wheel to go to a new column.
5. Press the navigation wheel or **ENTER** when selection is complete. The new channel is displayed.



After CLS (on the lower line of the display), the closed channels are listed. If no channels are closed, <none> is displayed here. If the list of closed channels extends past one screen, "..." is displayed at the end of the lower line. To see the full list of closed channels, press **DISPLAY** until the list of closed channels is displayed.

NOTE

For the Model 707B, also see [Selecting channels from the front panel](#) (on page 2-17) to select a channel.

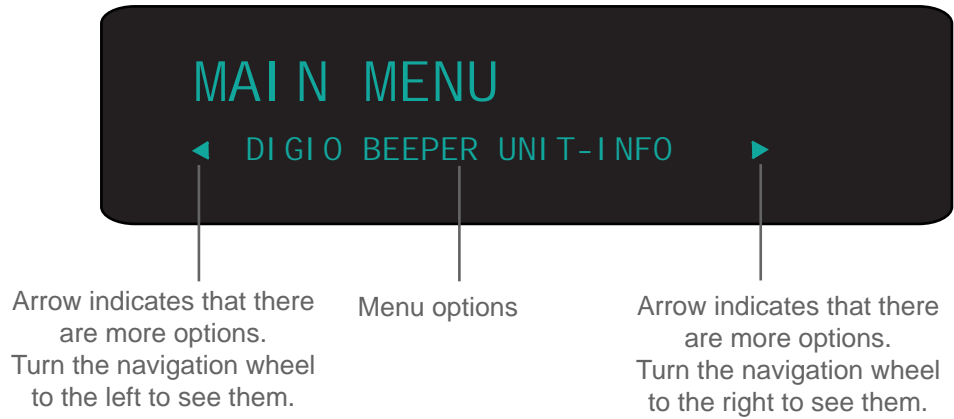
Display during setup

During setup, the display shows menu choices that you can use to configure the instrument.

To use the menus, you use the navigation wheel  to scroll through menu options. When a menu item is selected, it blinks. Press the navigation wheel  or **ENTER** to select an option.

In the following figure, the Main Menu is displayed, with arrows showing that there are additional menu items.

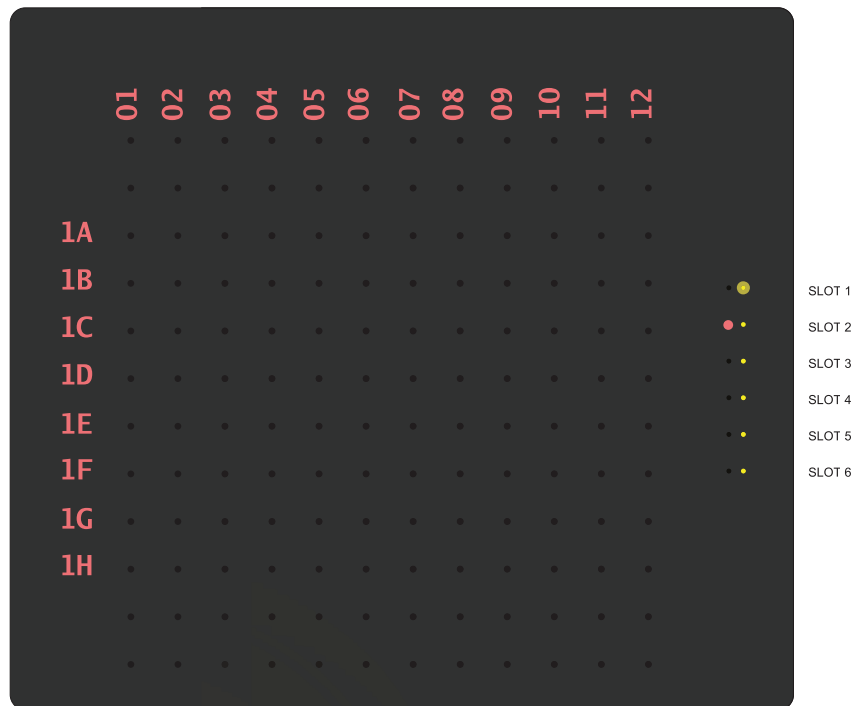
Figure 15: Front-panel Main Menu display



Crosspoint display (Model 707B only)

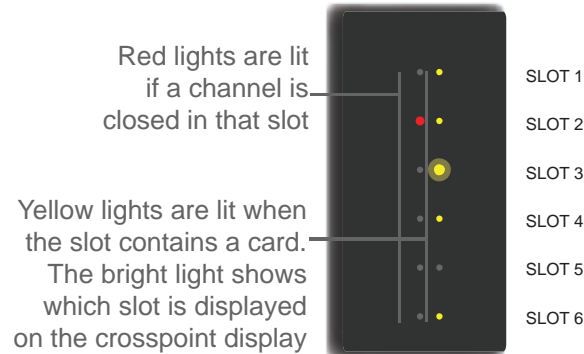
The crosspoint display on the front panel of the Model 707B displays information about the slots that contain cards and the open and closed state of the channels for one card slot at a time. If there are no cards in the instrument's slots, the crosspoint display shows "No card in unit."

Figure 16: Model 707B crosspoint display



The crosspoint display has a list of slots on the right. To the left of the slot list are lights that show you an overview of the cards in the instrument's slots.

Figure 17: Model 707B slot indicators



The red lights indicate closed channels. If a red light is on, a channel in that slot is closed. If the red light is not lit for a slot, all channels in that slot are open. In the figure above, the card in slot 2 has at least one closed channel.

The yellow lights indicate which slots contain a card and which slot is presently displayed on the crosspoint display. When a yellow light is on, the slot contains a card. When a yellow light is off, the slot does not contain a card. If the yellow light is brighter than the others, that slot is being displayed on the crosspoint display. In the figure above, there are cards in five slots, no card in slot 5, and the channels for the card in slot 3 are displayed on the crosspoint display.

Crosspoint display description

The crosspoint display displays all channels for one card slot at a time. If there are no cards in the instrument's slots, the crosspoint display shows "No card in unit."

The rows are labeled by default with the slot number following by the row number. For example, slot 1 would start with 1A, 1B and slot 2 would start with 2A, 2B and so on.

The columns are labeled by default as 01, 02, and so on.

If a red light is on at a row-column crosspoint, the channel at the crosspoint is closed. If no red light is on, the channel is open. Channels appear in different LED intensities when a row is selected. Channels that are closed appear brighter than ones that are presently selected.

You can change the label names; see [Set up labels](#) (on page 2-98).

You can also open and close channels by pressing **CHAN** to use the [Channel Action Menu options](#) (on page 2-22).

Selecting channels from the front panel

From the front panel, you can change the display to show another card slot, choose a specific channel, and open and close channels. For the 707B, the crosspoint display shows you the status of the channels for one card at a time.

To display a different slot:

Press the navigation wheel, then turn the navigation wheel to the right to go to the next slot or to the left to go to the previous slot. You must go through all of the rows on one slot to go to the next slot.

To choose a specific row:




1. When you are displaying the slot that contains the row, press the navigation wheel . The red lights for that row are displayed.
2. Turn the navigation wheel  to go to the row you want to select.
3. Press the navigation wheel  to select the row. The red lights for each crosspoint in the row are on, as shown in the following figure.




Figure 18: One row selected



NOTE

On the Model 707B, if you scroll past the last row, you will go to the next slot.

To choose a specific column in the selected row:


1. After choosing the row, press the navigation wheel . A column of red lights is displayed.
2. Turn the navigation wheel  to go to the column. Note that scrolling through the columns will not scroll through slots as scrolling through rows does.
3. Press the navigation wheel  to select the column and row. This channel is now displayed on the bottom display.

To open and close channels:

You can use the crosspoint display to view the open or closed status of a specific channel crosspoint.


After selecting the channel crosspoint so that the channel is displayed on the bottom display, you can press **OPEN** or **CLOSE** to open or close the channel.



Keys and navigation wheel


The keys and navigation wheel  on the front panel allow you to turn on, set up, and operate the instrument from the front panel.

The **POWER** key. Press this key to turn the instrument on (I). Press it again to turn the instrument off (O).






Navigation wheel

Turn the navigation wheel  to scroll to a menu option or to change the selected value.

Push the navigation wheel  to open menus or to select a menu option or a value. In most cases, pressing the navigation wheel  performs the same action as pressing the **ENTER** key.


On the Model 707B, you can use the navigation wheel  to control which slot is displayed on the crosspoint display.

To change a value with multiple characters:

1. Turn the navigation wheel  to go to the character you want to change (the character blinks when selected).
2. Press the navigation wheel  to edit that character.
3. Turn the navigation wheel  to change the value.
4. Press the navigation wheel  to keep the change.
5. Repeat these steps as needed to change the value.
6. Press the **ENTER** key or the navigation wheel  when finished changing all the characters.

Front-panel keys

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 7-82).

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 through the list of closed channels.

The **CONFIG** key accesses attribute menus in which you can configure channels and scans.

CONFIG and then **CHAN** opens the Channel Attribute Menu.

CONFIG and then **SCAN** opens the Scan Attribute Menu.

Keys

The top row of keys under the display allows you to open and close channels, work with scan lists, and load and run scripts.

Figure 19: Models 707B and 708B top row of keys



Key descriptions	
Key	Description
OPEN ALL	Opens all closed channels.
STEP	If a scan list has been defined, press STEP to step through the list. Each press is one scan step. See Basic scan procedure (on page 3-5).
OPEN	Opens the selected channel or channel pattern.
CLOSE	Closes the selected channel or channel pattern.
LOAD	Loads code or scripts that can be run from the front panel.
RUN	Runs the last code or script selected through the LOAD key.
INS	Appends the selected channel or channel pattern to the scan list.
DEL	Deletes the first occurrence of the selected channel or channel pattern from the scan list.


Also see:

- For detail on using **OPEN ALL**, **STEP**, **OPEN**, and **CLOSE**: [Closing and opening channels](#) (on page 2-94).
- For detail on using **LOAD** and **RUN**: [Load Test menu options](#) (on page 2-21).
- For detail on using **INS** and **DEL**: [Front-panel scanning](#) (on page 3-6).

The bottom row of keys allow you access menus and set up channels, patterns, cards, scans, triggers, and general instrument operation.

Figure 20: Bottom row of keys



Key descriptions	
Key	Description
CHAN	If a channel is displayed, opens the Channel Action Menu options (on page 2-22), which allows you to open and close channels. If a pattern is displayed, pressing CHAN switches to channel view.
PATT	If a pattern is displayed, opens the Pattern Action Menu options (on page 2-23), 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	Displays information about the installed cards and the instrument. Information includes the firmware revision, model name, and model number.
SCAN	Opens the Scan Action Menu options (on page 2-23), which allows you to run, manage, view, and reset scan lists. See Scanning and triggering (on page 3-1).
TRIG	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 7-84).
MENU	Opens the Main Menu options (on page 2-23), 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	This key: <ul style="list-style-type: none"> • Cancels the current selection and returns to the previous menu item. • Exits remote operation so you can control the instrument from the front panel. • Aborts a scan that is running. • Aborts a script that is executing.
ENTER	Accepts the current selection or brings up the next menu option. In most cases, pressing ENTER is the same as pressing the navigation wheel  .

LAN status indicator

The LAN status indicator 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 indicator blinks when you click the **ID** button in the upper right corner on the home page.

See [LAN communications](#) (on page 2-36) for more information.



Set beeper and key clicks

You can turn the instrument beeper and key click sounds on or off.

NOTE

Disabling the beeper also disables the keyclicks. To enable keyclicks, you must also enable the beeper.

To change the beeper or key click sounds from the front panel:

1. Press **MENU**.
2. Use the navigation wheel  to select BEEPER.
3. Select KEYCLICK or BEEP.
4. Select ENABLE or DISABLE.
5. Press **ENTER** or press the navigation wheel  to save the change.
6. Press **EXIT (LOCAL)** to return to the Main Menu.

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.

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

The User option loads code that was added to Load Test with the [display.loadmenu.add\(\)](#) (on page 7-74) command.

The Scripts option loads named scripts that were added to the runtime environment. See [Manage scripts](#) (on page 6-3) 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 Attribute Menu options

The options in the Channel Attribute Menu allow you to configure channels from the front panel.

To open the Channel Attribute Menu, go to channel view. Select the channel for which you want to set attributes, then press **CONFIG**, then press **CHAN**.

The options are:

- **LABEL:** Sets the label that is displayed on the front panel for the specified channel.
- **LABEL-ROW:** Sets the label that is displayed on the front panel for the specified row.
- **LABEL-COL:** Sets the label that is displayed on the front panel for the specified column.
- **FORBID:** Allows you to prevent the channel from being closed.
- **DELAY:** Sets delay time (in addition to settling time) for the specified channels. Enter the value for the delay in 1ms steps from 0 to 60 seconds for a channel.
- **COUNT:** Displays closure cycles for the specified channel.

For more information about channel attribute settings, see [Channel attributes](#) (on page 2-97).

Scan Attribute Menu options

Use the options in this menu to configure scans from the front panel.

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

Options include:

- **ADD:** Reminder that you need to use **INS** to add channels or channel patterns to a scan.
- **BYPASS:** Allows you to bypass the trigger for the first step of the first scan. See [Trigger model](#) (on page 3-1) for more information.
- **MODE:** Selects how the scan initializes the instrument when the scan is executed. Choose OPEN-ALL to open all channels or OPEN-SELECT to open only those involved in the scan.
- **SCAN_CNT:** Sets the scan count, which is the number of times that the instrument repeats the steps in a scan. After repeating the steps this number of times, the instrument returns to idle.

For information on configuring scans, see [Changing attributes of an existing scan](#) (on page 3-6).

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**.

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.

For more information, see [Working with channels](#) (on page 2-87).

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 2-99).

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 INS 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.

Main Menu options

The options in the Main Menu allow you to create a configuration script, set up communications, verify and set some instrument operation, set up digital input/output, and get instrument information.

To open the Main Menu, press **MENU**.

Main Menu options

Option	Description	Also see
SCRIPT	Option for creating a script that stores the present configuration of the instrument.	Save the present configuration (on page 2-106)
GPIB	Options for setting up GPIB communications.	GPIB (on page 2-31)
DDC	Options that allow you to run existing 707A or 708A applications.	Using Models 707A and 708A compatibility mode (on page B-1)
LAN	Options for setting up LAN communications.	LAN communications (on page 2-36)
TSPLINK	Options for configuring TSP-Link	TSP-Link system (on page 6-45)
CHANNEL	Select a connection rule to determine the order in which switch channels are opened and closed, and select whether to connect sequentially.	Connection methods for close operations (on page 2-89)
DISPLAY	Verify operation of the keys, display, and crosspoint display LEDs.	Testing the display, keys, and channel matrix (on page 8-9)
DIGIO	Options for controlling the digital input and output lines.	Digital I/O port (on page 2-7)
BEEPER	Enables or disables the instrument key clicks and beeps.	Set beeper and key clicks (on page 2-21)
UNIT-INFO	Displays the firmware version, serial number, memory usage, and fan status. Fan status is 707B only.	Check fan status (on page A-2)
RESET	Resets the instrument.	reset function (on page 5-13)

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)

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.

USB

To use the USB connection, you must have the Virtual Instrument Software Architecture (VISA) layer on the host computer. See [VISA](#) (on page 2-58) and [How to install the Keithley I/O Layer](#) (on page 2-61) for more information.

VISA contains a USB Class driver for the USB Test & Measurement Class (USBTMC) protocol which, once installed, allows the 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" section refers to devices that implement the USBTMC or USBTMC-USB488 protocol.

NOTE

The full version of National Instruments VISA provides a utility to create a USB driver for any other kind of USB device with which you might want to communicate with VISA. For more information, see the [NI VISA website](http://www.ni.com) <http://www.ni.com>.

Connecting USB instruments to the computer

To connect the instrument to the computer with USB, you will use a USB cable with a USB Type B connector on one end and a USB type A connector on the other end.

To connect the USB cable to the computer:

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

Communicate with the instrument

To communicate with the USB device you need to use VISA. VISA requires a resource string to connect to the correct USB instrument of the format:

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

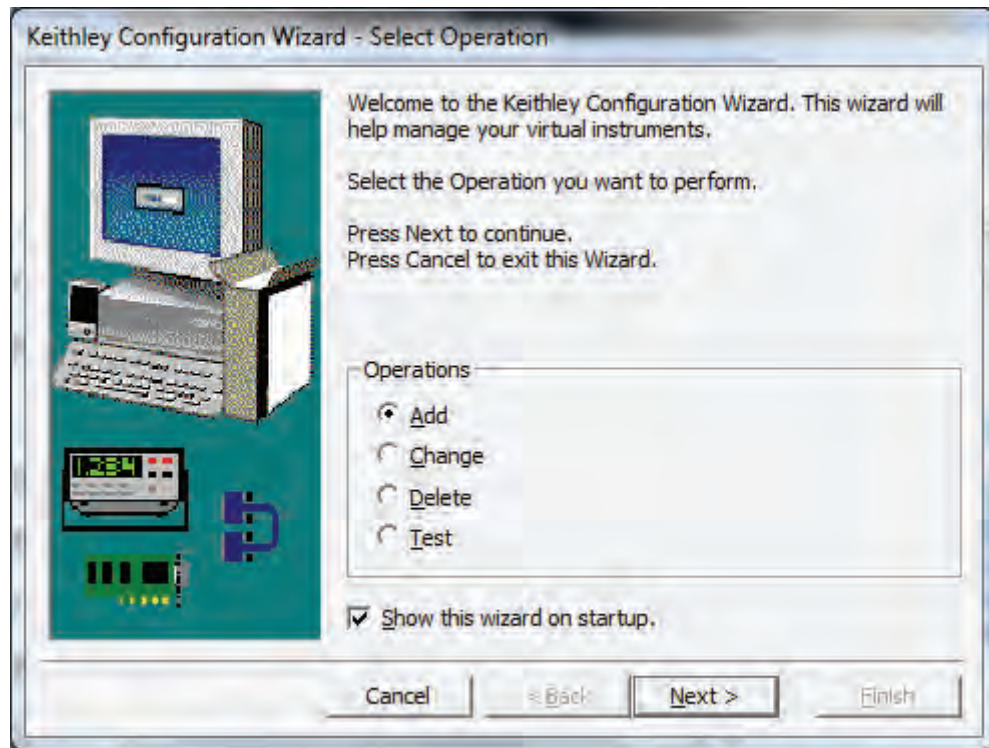
This requires you to determine the parameters. You can gather this information by running a utility that automatically detects all instruments connected to the computer.

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

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

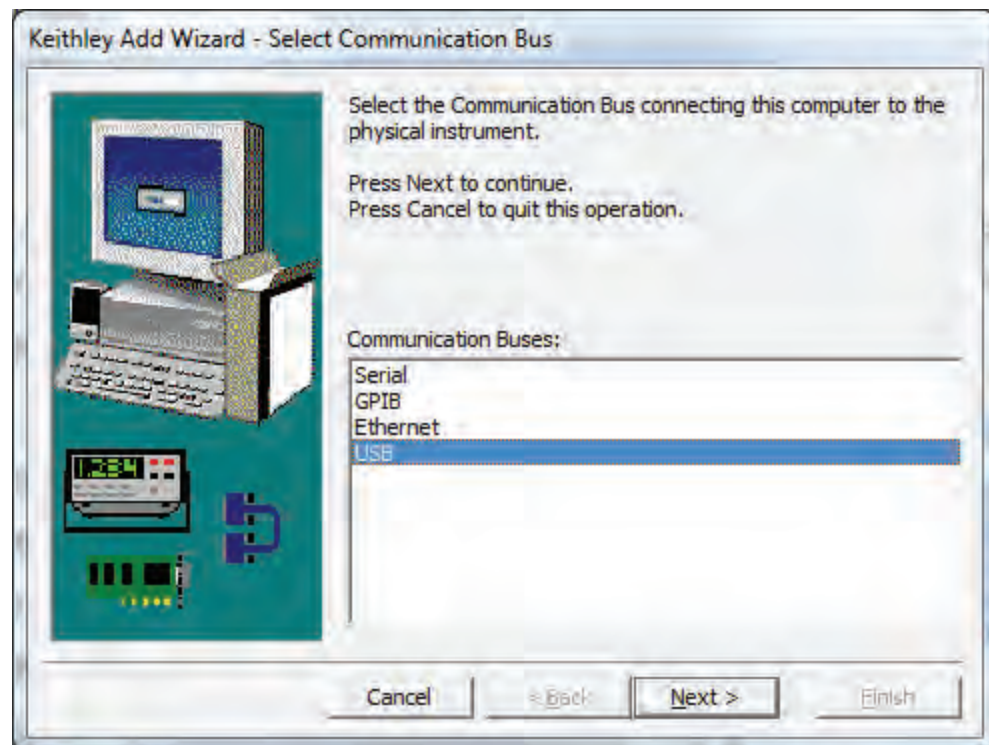
1. Start the Keithley Configuration Panel. The Select Operation dialog box is displayed.
2. Select **Add**.

Figure 21: Select Operation dialog box



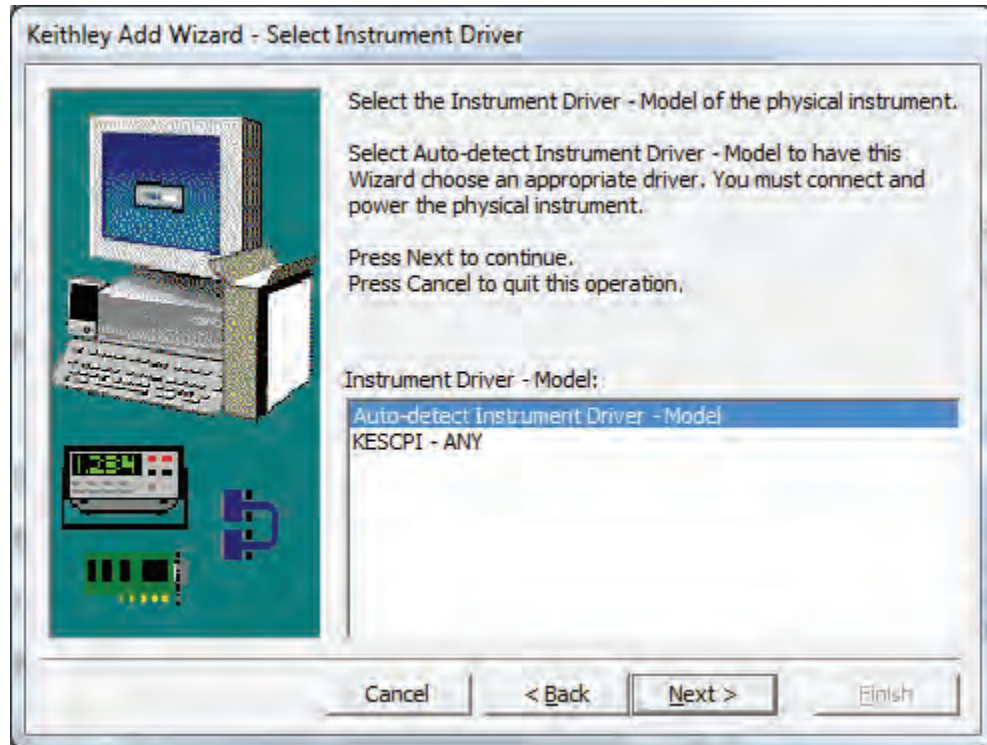
3. Click **Next**. The Select Communication Bus dialog box is displayed.

Figure 22: Select Communication Bus dialog box



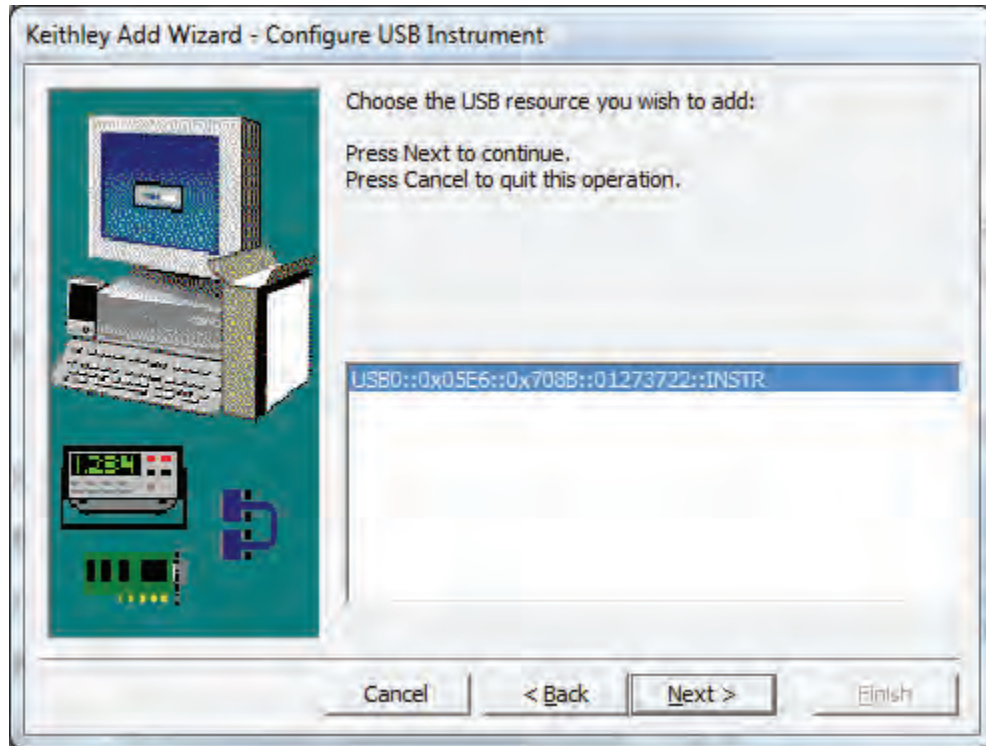
4. Select **USB**.
5. Click **Next**. The Select Instrument Driver dialog box is displayed.

Figure 23: Select Instrument Driver dialog box



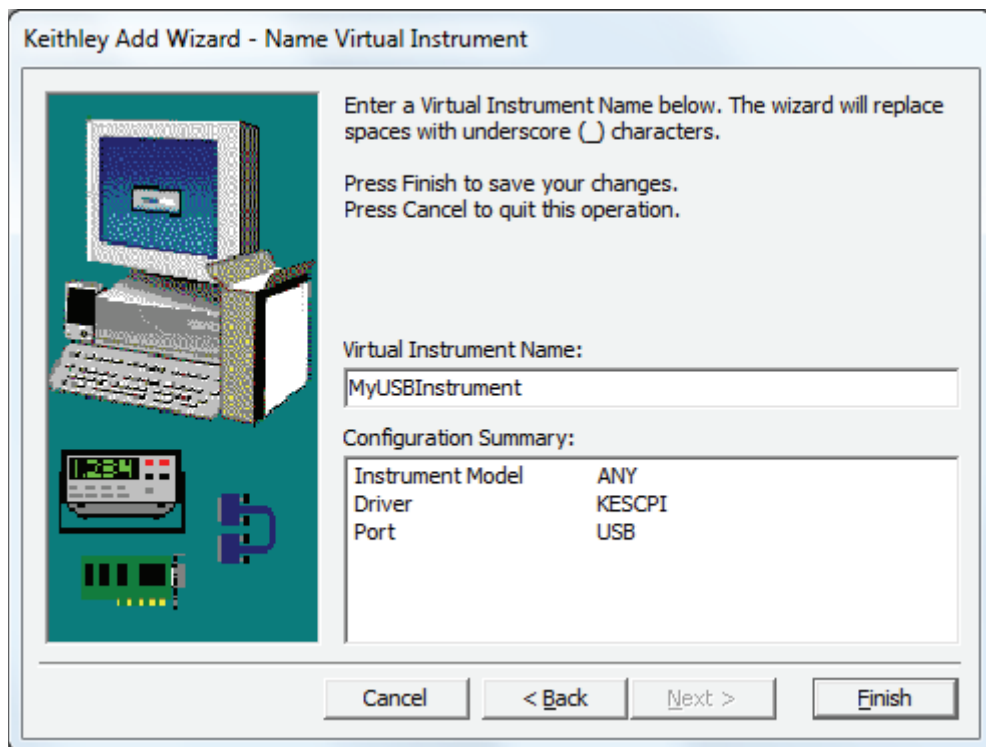
6. Select **Auto-detect Instrument Driver - Model**.
7. Click **Next**. The Configure USB Instrument dialog box is displayed with the detected instrument VISA resource string displayed.

Figure 24: Configure USB Instrument dialog box



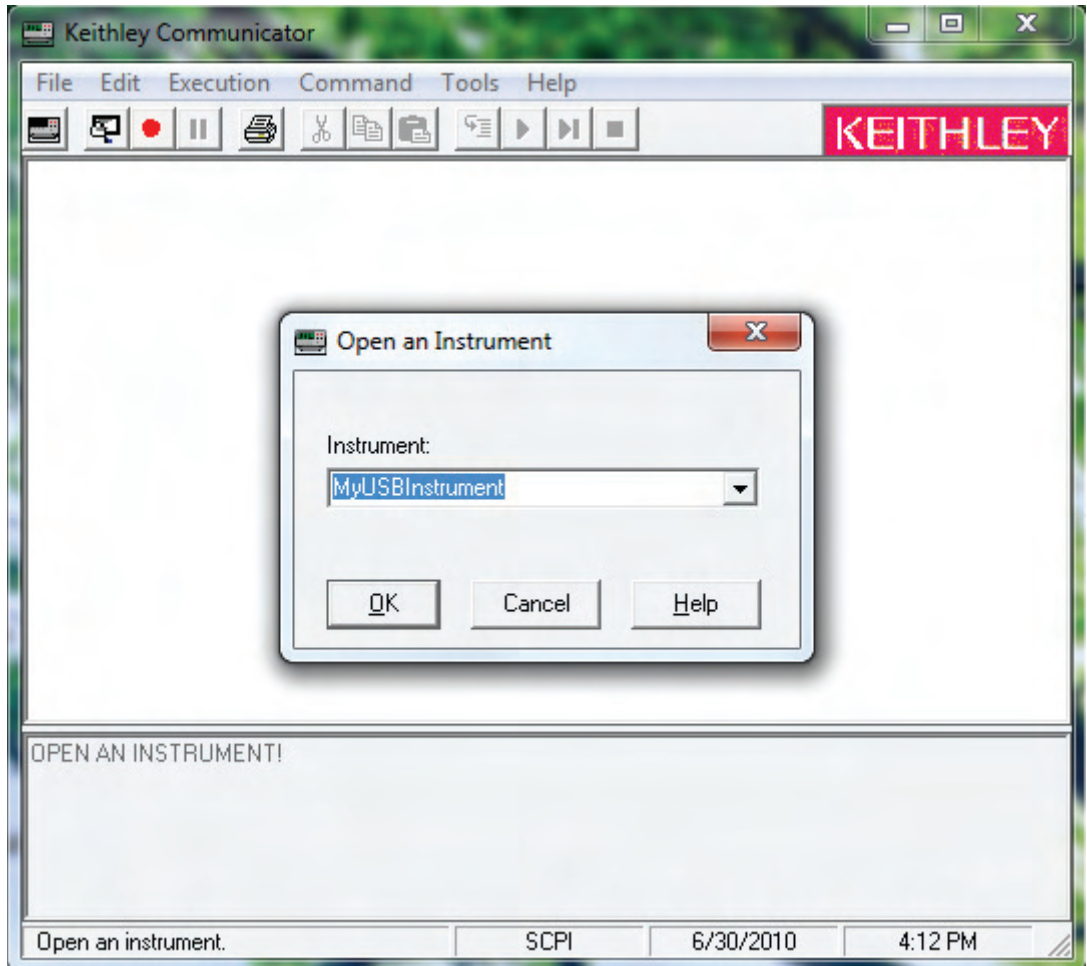
- 8. Click **Next**. The Name Virtual Instrument dialog box is displayed.

Figure 25: Name Virtual Instrument dialog box.png



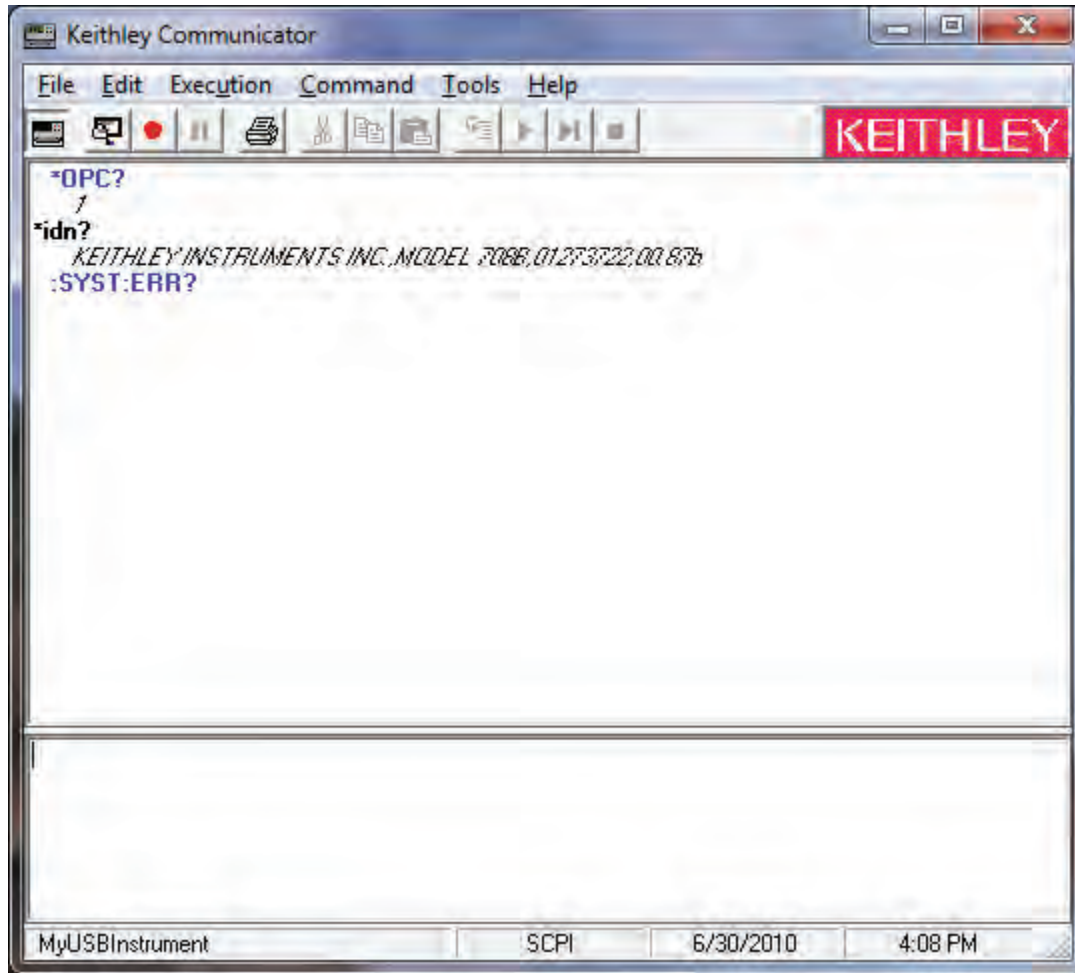
9. In the Virtual Instrument Name box, enter a name that you want to use to refer to the instrument.
10. Click **Finish**.
11. Click **Cancel** to close the Wizard.
12. Save the configuration. From the Configuration Utility, select **File > Save**.
13. In the Keithley Communicator, select **File > Open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

Figure 26: Keithley Communicator Open Instrument



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

Figure 27: Send a command to the instrument



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, either of which can be used with Model 707B or 708B:

- **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).

USB VISA identifiers

The USB identifiers to communicate with your Keithley instrument using VISA are:

- **707B:** USB0::0x05E6::0x707B::[serial number]::INSTR
- **708B:** USB0::0x05E6::0x708B::[serial number]::INSTR

Where:

- USB0: USB interface
- 0x05E6: The Keithley vendor ID (assigned to Keithley Instruments by the USB.org)
- 707B or 708B: Instrument model number
- [serial number]: The serial number of the instrument (the serial number is on the rear panel)
- INSTR: Use the USBTMC protocol

GPIB

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

GPIB standards

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

GPIB quick start

Install the GPIB driver software

Check the documentation for your GPIB controller for information on where to acquire drivers. We also recommend that you check the vendor's website for the latest version of drivers or software.

It is important to install the drivers before the hardware so that the incorrect driver does not get associated with the hardware.

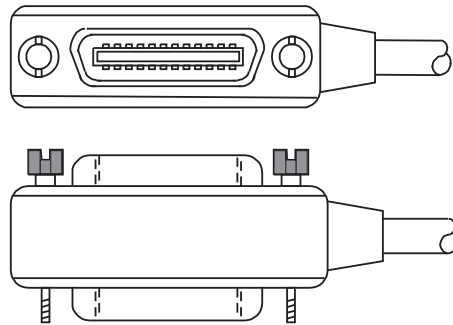
Install the GPIB cards in your computer

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

Connect the GPIB cable

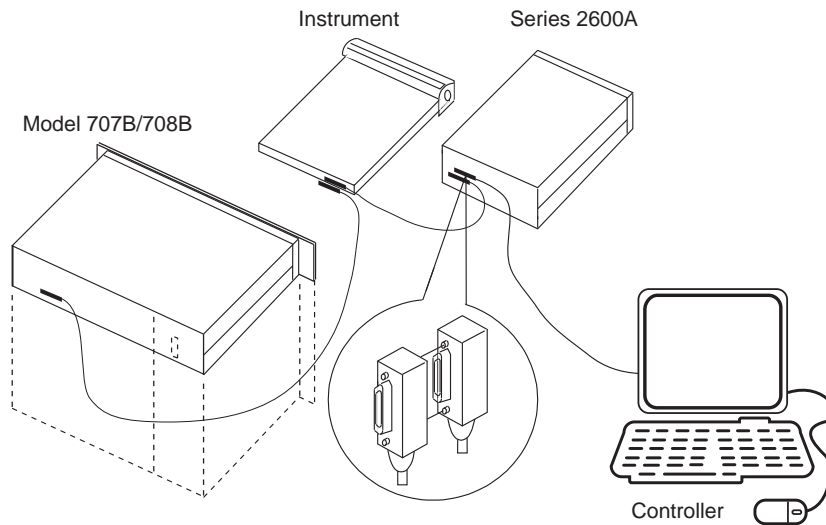
To connect a Model 707B or 708B to the GPIB bus, use a cable equipped with standard IEEE-488 connectors, as shown below.

Figure 28: IEEE-488 connector



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

Figure 29: IEEE-488 connections

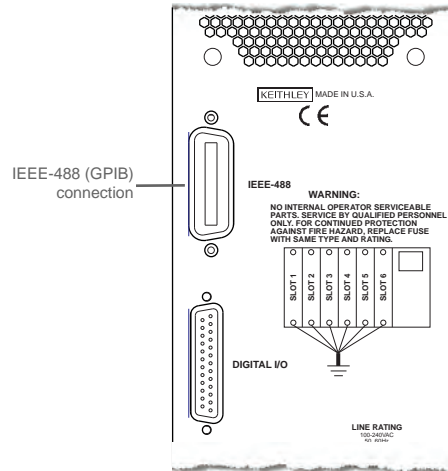


CAUTION

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 Model 707B or 708B 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 30: Model 707B rear panel IEEE-488 connection



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

Figure 31: Model 707B rear panel IEEE-488 connection

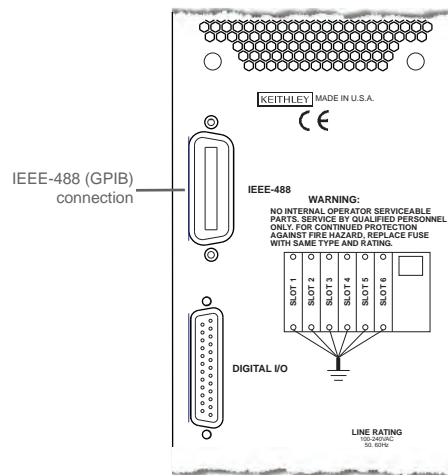
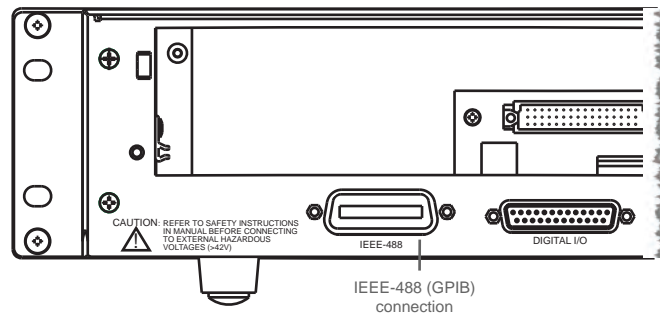


Figure 32: Model 708B rear panel IEEE-488 connection

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 the address 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 7-140) 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 7-97) 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 2-61).

Terminator

When receiving data over the GPIB, the instrument terminates on any line feed character or any data byte with EOI asserted (line feed with EOI asserted is also valid). When sending data through the GPIB drivers, it is not necessary to append a line feed character to all outgoing messages. The EOI line is asserted with the last character sent.

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 just changing the resource string in the VISA Open method.

GPIB reference

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 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 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 configure the controller. Do not forget to save your settings.

LAN communications

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.

Models 707B and 708B are class C LXI version 1.2 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 Models 707B and 708B 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.

LAN quick start

Overview

This section describes how to connect an instrument directly to a computer using a LAN connection. To do this, you will:

1. Identify and record network settings.
2. Configure the network settings on the computer and instrument.
3. Create a direct instrument-to-PC LAN connection.
4. Set up an IP address between the computer and the instrument.
5. Access the instrument's internal web interface.

Configure the computer's network interface card to obtain an IP address automatically

NOTE

Do not change your IP address without consulting your system administrator. Entering an incorrect IP address can prevent your computer from connecting to your corporate network.

Identify and record the existing IP configuration**CAUTION**

You are responsible for returning settings to their original configuration before reconnecting the computer to a corporate network. Failure to do this could result in damage to your equipment or loss of data. These settings include, but are not limited to, the IP address, DHCP enabled mode, and the subnet mask.

Record the existing IP configuration information (for the PC) in the table below so that you can return all settings back to their original configuration before reconnecting your computer to a corporate network.

DHCP Enabled	
IP Address	
Subnet Mask	
Default Gateway	
DNS Servers	

Set up automatic IP address selection***To set up automatic IP address selection:***

1. On the host computer, open the Internet Protocol Properties dialog box. See [Windows network configuration settings](#) (on page 2-48) for instructions on opening this dialog box.
2. Select **Obtain an IP address automatically**.
3. Click **OK**. Close the network settings dialog boxes.

Configure the instrument to obtain an IP address automatically

1. From the front panel of the instrument, press the **MENU** key.
2. Use the navigation wheel to select **LAN > CONFIG > METHOD**. Press the navigation wheel to display the METHOD menu.
3. Select **AUTO**. Press the navigation wheel to select AUTO and return to the LAN CONFIG menu.
4. Press the **EXIT** key once to return to the LAN MENU.
5. Select **APPLY**. Press the navigation wheel to apply the setting. The Main Menu is displayed.

Connect the LAN cable

Connect the LAN 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.

The location of the LAN connector on the instrument is shown below.

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.

Figure 33: Model 707B rear panel LAN connection

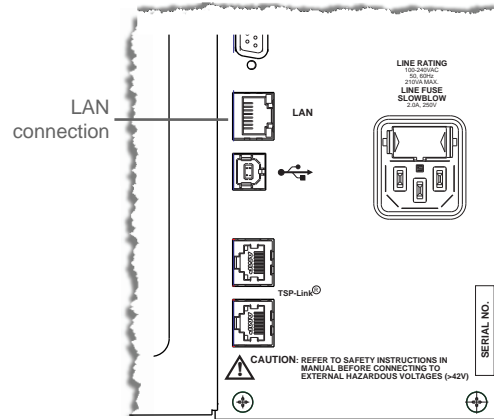
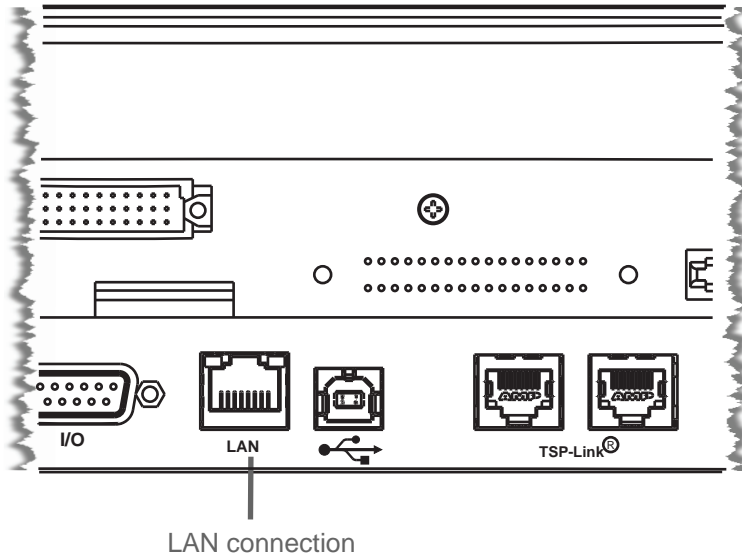


Figure 34: Model 708B rear panel LAN connection



Wait for the LAN Status indicator on the front panel to turn solid green

The LAN status indicator confirms that the Model 707B or 708B has been assigned an IP address. Note that it may take several minutes for the computer and Model 707B or 708B to establish a connection.

Install LXI Discovery Browser software on your computer

You can use the LXI Discovery Browser to identify the IP addresses of LXI certified instruments that are set up for automatic IP address selection. Once identified, you can double-click the IP address in the LXI Discovery Browser to open the web interface for the instrument.

The LXI Discovery Browser is available on the instrument CD. It is also available on the [Keithley website](http://www.keithley.com) (<http://www.keithley.com>).

To locate the LXI Discovery Browser on the website:

1. Select the **Support** tab.
2. In the model number box, type **707B** or **708B**.
3. From the list, select **Software**. A list of software applications for the Model 707B or 708B is displayed.
4. See the readme file included with the application for more information.

For more information on the LXI Consortium, see the [LXI Consortium website](http://www.lxistandard.org) (<http://www.lxistandard.org>).

Run LXI Discovery Browser

The software populates a dialog box with the instruments found on the network and their associated IP addresses.

Double-click the IP address in the LXI Discovery Browser dialog box. The instrument web page opens.

For information on using the web page, see [Using the web interface](#) (on page 2-69).

LAN reference

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 1.2 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.

Instrument LAN protocols

Raw socket communications

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. There is no extra protocol overhead above and beyond what TCP gives you. A programmer can easily communicate with the instrument using Winsock on Windows computers or Berkley sockets on Linux or Apple computers.

The port number to use for connections differs with instrument models:

- 707B and 708B: Port 5025

Dead socket connections

If a computer is connected to an instrument through TCP and the computer application is terminated without releasing the socket, it can leave the port on the instrument “hanging”. You cannot reconnect to it without switching the power to the instrument off and then back on.

To avoid cycling power when this occurs, some instruments have a dead socket port (sometimes known as a backdoor). The dead socket termination port is used to terminate all existing LAN connections. 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 LAN connections are terminated and closed when the connection to the dead socket termination port is closed.

The dead socket termination port for Models 707B and 708B is 5030. Connect to this port, and then when you disconnect, the dead port will be cleaned up (released)

VXI-11

VXI-11 is a LAN protocol that emulates GPIB over Ethernet. It uses remote procedure calls to call functions in the instrument for creating a link, sending data, reading data, and so on. There is also a small header that indicates how much data is being sent. This means there is some overhead added. Therefore VXI-11 is slower than raw socket communication. On the other hand, with VXI-11, the programmer or driver writer does not have to confirm that the correct number of bytes have been sent and received.

VXI-11 also supports an out-of-band channel, which allows the instrument to signal to the computer that an event, such as a SRQ, has occurred.

VXI-11 has a limitation in that it uses broadcast packets to locate the instrument when it wants to make a connection.

Models 707B and 708B use port 1024 for VXI-11 communication. You do not have to know this port number to connect through VXI-11. The discovery portion of the protocol will negotiate the port number for you. If you are trying to configure a firewall, this port number might be useful.

LAN network types

Ethernet is a type of Local Area Network (LAN) that works with a variety of transmission media. Some of the more popular variations are 10/100BaseT, 10Base2, and 10BaseF, which use unshielded twisted pair (UTP), coaxial cable, and optical fiber, respectively.

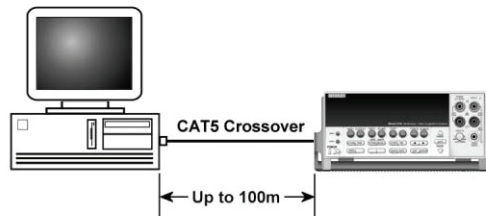
Most of Keithley's instruments work with a 10/100 BaseT network and use a standard RJ-45 connector. This is an eight-wire connector, but only four wires are used: one pair to transmit and one pair to receive data. A 10BaseT network can accommodate transmission speeds up to 10Mbps/second; 100BaseT operates at up to 100Mbps/second. Both types of networks usually require Ethernet hubs to make connections. The exception is a one-to-one connection using a crossover cable (see below).

The LAN connector on an instrument gives you more flexibility than GPIB and RS-232 interface controller-subordinate configurations. Rather than connecting the instrument directly to a computer controller in a closed loop, a LAN instrument can be connected to a TCP/IP network using its own subnetwork, or it can be connected directly to an existing network, including a corporate intranet.

One-to-one connection

A one-to-one connection using a network crossover cable connection is similar to a typical RS-232 hookup using a null modem cable. The crossover cable has its receive (RX) and transmit (TX) lines crossed to allow the receive line input to be connected to the transmit output on the network interfaces. With most instruments, this is only done when a single instrument is being connected to a single network interface card.

Figure 35: One-to-one connection with a crossover cable



NOTE

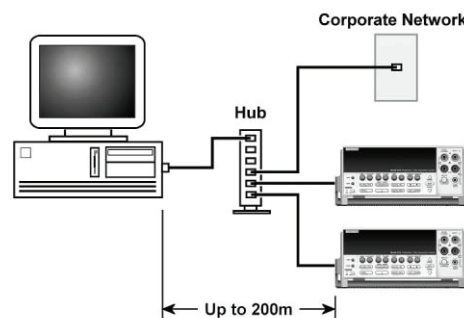
The Models 707B and 708B support Auto-MDIX and can use either normal Ethernet CAT-5 cables (patch) or crossover cables. The instrument automatically adjusts to support either cable.

One-to-many connection

With an Ethernet hub, a single network interface card can be connected to as many instruments as the hub can support. This requires straight-through network (non-crossover) cables for hub connections.

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

Figure 36: One-to-many connection using a network hub or switch

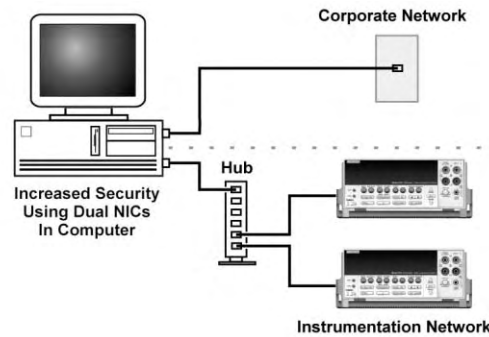


Use two network interface cards to connect to a corporate network and instrumentation hub

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

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

Figure 37: Use two network interface cards to connect to a corporate network and instrumentation hub

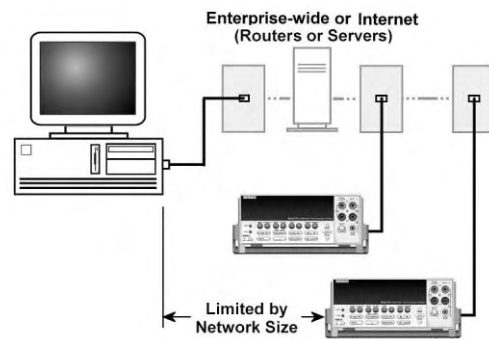


Instrumentation connection to enterprise routers or servers

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

Usually, the instruments are kept inside the corporate firewall, but the network administrator could assign resources that allow them to be outside the firewall. This would allow instruments to be connected to the Internet using appropriate security methods. Thus, data collection and distribution could be controlled from virtually any location.

Figure 38: Instrumentation connection to enterprise routers or servers



Setting up an isolated instrument network

The following describes how to set up a simple isolated Class C network for communicating with two LAN instruments using static IP addressing. This network example is similar to the network shown in [One-to-one connection](#) (on page 2-42), but without the corporate network connection to the hub.

The standard Ethernet hub basically repeats anything it receives from one port, making that data available to all its other ports. Hub connections are made with straight-through cables. The hub is connected to the network interface card in the computer. The network interface card and its driver must be properly installed on the computer according to the manufacturer's instructions. You can also use a switch; the benefit of a switch is that it does not forward network packets out all of the ports. It only forwards those that are being addressed by the packet.

To set up the network:

1. Create IP addresses for the three hosts (the network interface card and two instruments) on the network. This is a Class C network, so the subnet mask is 255.255.255.0. From the table in [IPv4 address syntax and subnets](#) (on page 2-46), note that the first three parts of the IP address make up the network ID. For purposes of this example, a network ID of 192.68.1 is used, which is the default network ID that is shipped with the most of Keithley's instruments.

If a corporate network is also connected to the same computer using dual network interface cards, the instrumentation network ID must be different than the corporate network ID.

2. Assign the host ID portions of the three IP addresses. In this example, a host number of 1 is assigned to the network interface card. The first instrument is assigned a host number of 10 and the second instrument becomes host number 20. The complete IP addresses are listed below.

Example host IP address

Card or instrument	IP address
Network interface card	192.68.1.1
First instrument	192.68.1.10
Second instrument	192.68.1.20

3. In a Windows operating system, use the Windows Control Panel to assign the network interface card IP address. The exact steps differ somewhat for each version of Windows. See [Windows network configuration settings](#) (on page 2-48).
4. Using the front-panel menus, assign a unique IP address to each of the other two instruments. See [Set the IP address on the instrument](#) (on page 2-45).

NOTE

It is a good idea to record IP addresses for reference. This is especially important when changing the existing network settings on the computer.

5. Verify that the instruments and the network have been set up and are working properly. You can try logging into the web interface of the instrument to test the connection. See [Connect to the instrument web interface](#) (on page 2-69).
6. If you are unable to establish communications, double-check the network settings and try again.

To set the IP address:

1. From the front panel, press the **MENU** key.
2. Use the navigation wheel to select **LAN > CONFIG > METHOD**. Press the navigation wheel to display the METHOD menu.
3. Select **MANUAL**. Press the navigation wheel to select your choice and return to the LAN CONFIG menu.
4. Select **IP_ADDRESS**. Press the navigation wheel to display the current IP address.
5. If you are:
 - Connecting directly to a host computer, set all but the last three digits to match the IP address of the host computer. Change the last three numbers (after the last decimal point) to a number that is unique on the LAN. The last three digits may be anything from 1 to 254 for a subnet mask of 255.255.255.0.
 - Connecting to a network: Enter the address provided by your system administrator.

**Quick Tip**

To set the address, turn the navigation wheel to go to the number that needs to change, then press the navigation wheel. Turn the navigation wheel to change the number, then press the navigation wheel to set that number. Repeat this for each number that needs to change.

6. Record the instrument's IP address.
7. Press the **ENTER** key when the IP address is complete. The LAN CONFIG menu is displayed.
8. Press the **EXIT (LOCAL)** key once to return to the LAN menu.
9. Select **APPLY**. Press the navigation wheel to save the change. The Main Menu is displayed.

TCP/IP network basics

Regardless of the type of network connection used, there must be a way to identify each network device on a network. A software driver installed in the computer provides the means of controlling the instrument. A data communication protocol defines the method of exchanging instructions and data between the computer and each instrument.

**CAUTION**

When connecting to a corporate network, the network administrator **MUST** provide all of the network settings for the LAN instrument. Failure to use settings provided by the network administrator could result in failures at other locations on the corporate network. Failure to work through the network administrator could also be considered a breach of company policy. Always consult with the network administrator before attempting to connect instrumentation to the network.

LAN and LXI instruments use the TCP/IP protocol to communicate with other hosts on the network. A host is defined as any device on the network that can transmit and receive IP packets. In addition to the instrumentation, this includes workstations, servers, and routers. Each host on a TCP/IP network is assigned an IP address that is unique to that host.

IPv4 addressing

Models 707B and 708B support IPv4 addressing. There are three ways of assigning an IP address to a host: DHCP, DLLA, and static.

DHCP

The Dynamic Host Configuration Protocol (DHCP) protocol is a way for network devices to request Internet protocol (IP) parameters such as the IP address, gateway, subnet mask, and DNS server addresses. Each time a network device connects, it leases the IP address for an amount of time set by the DHCP server. Generally, the time is 24 hours, but it could be shorter or longer depending on how the DHCP server was configured.

Typically, DHCP IP addressing is used for corporate networks. It is also commonly the default on home wireless routers that you use to connect your home computer to the internet.

One of the benefits of DHCP is that a system administrator does not have to give you an IP address each time you want to connect something to the network. However, one of the drawbacks is that you do not know the IP address that may be assigned to your network device and it can change from connection to connection. While there are ways to configure DHCP servers to always give the same network device the same IP address each time, but that means a system administrator needs to be involved.

Dynamic Link-local Addressing (DLLA)

Also called Auto-IP, DLLA was originally used for ad-hoc networks. DLLA allows all the network devices to automatically allocate their own unique IP addresses. An Auto-IP address is in the range 169.254.0.0 —169.254.255.255. A network device randomly picks an address in this range and sends out an ARP packet (on IPv4 only) to see if any other device is using it. If another device responds that it is using it, the network device generates another address and sees if it is in use and so on.

Static or fixed IP address

Static IP addressing means that the person setting up and configuring the network has assigned a fixed, unique IP address to each network device. This requires more rigorous enforcement to make sure everyone has a unique IP address but it has the added benefit that the address will not change.

IPv4 address syntax and subnets

For IPv4, the IP address is 32 bits wide and is divided into two main parts: a network ID number and a host ID number. The address is expressed as four decimal numbers separated by three periods. Valid addresses range from 0.0.0.0 to 255.255.255.255, for a total of about 4.3 billion unique addresses. Each of the four numbers represents the decimal value of the numbers' 8-bit bytes. The way these four numbers are assigned for host ID and network ID depends on the class of network being used.

The network ID must be unique among all network subnets that connect to the Internet or corporate intranet. If the subnet will be connected to the public Internet, the network ID must be obtained from the Network Information Center, which assigns and preserves unique IDs. In any case, each host ID must be unique among all the hosts on the same network (which presumably has a unique network ID number).

In the TCP/IP protocol, a subnet mask separates the network ID from the host ID. The subnet mask looks like an IP address, but sets a data bit high for each position of the IP address that makes up the network ID. Three different classes of network are defined with the IP address and subnet mask, as shown in the following table.

Network classes defined by IP address and subnet mask combinations

Network class	IP address	Subnet mask	Available subnets	Available hosts
A	nnn.hhh.hhh.hhh	255.0.0.0	126	16777214
B	nnn.nnn.hhh.hhh	255.255.0.0	16384	65534
C	nnn.nnn.nnn.hhh	255.255.255.0	2097151	254

NOTE

In the IP address format, "n" is a network ID position, and "h" is a host ID position. For simplicity, the first byte definition has been omitted from the table.

Class C networks are the most common and use the subnet mask 255.255.255.0. The first three bytes are the network ID number and the last byte is the host ID on the network. Host ID numbers 1 through 254 are available for assignment. All hosts on the same isolated network must have the same subnet mask. As a general rule, the top and bottom host numbers are reserved. The top one (nnn.nnn.nnn.255) is the broadcast address and the bottom one (nnn.nnn.nnn.0) is shorthand for the whole subnet.

DNS

The domain naming system (DNS) is a protocol that provides a way to associate a user-friendly name to an IP address. For example, while few people know the IP address of Google's website, everyone knows www.google.com. When you enter this URL into an Internet browser, the DNS on the network looks up the URL and translates it to the IP address for the Google website. This is invisible to the user.

For DNS to work, there must be a DNS server on the network and the correct IP address for that server must be configured in the computer. Some LAN instruments support DNS — if so, the IP address for the DNS server must be configured as well as the instrument IP address. Instruments, especially LXI instruments, also have to show a valid hostname on their LXI LAN Welcome and IP Configuration pages, and they need to use the DNS to validate that any hostname they display is valid. If they fail to validate a hostname, they must display the IP address for the instrument or a blank hostname in the hostname field on the web page.

DNS requires a network administrator that can update the database in the DNS server with any host name and IP address combinations, so it is not usually suitable for instrumentation setups.

Dynamic DNS

DNS is a rigid and inflexible system, because you must have a system administration add the DNS entry to the DNS server for it to work. Dynamic DNS tries to address this inflexibility. It addresses the needs of network devices that are powered up and down with IP addresses that can change several times a day.

Dynamic DNS is generally used to refer to a system where there is a DHCP server on the network that allocates IP addresses to the network devices, and therefore there is a method to register a hostname with the DHCP server. The DHCP server assigns the IP address and tracks the hostnames at the same time.

Multicast DNS

Multicast DNS (mDNS) is a protocol that is more suitable than DNS for small localized ad-hoc networks.

mDNS uses multicast packets for network devices to inform each other of their IP addresses, hostnames, and advertise what services might be available on that device. The packets are usually blocked from going any further than the nearest router. This limits the scope of mDNS, but multicast packets are more network-friendly than broadcast packets. For example, every network device in your subnet receives broadcast packets regardless of need, while for multicast packets, a network device must register for multicast addresses that will be received.

Multicast packets have IPv4 addresses in the range 224.0.0.0 through 239.255.255.255.

Windows network configuration settings

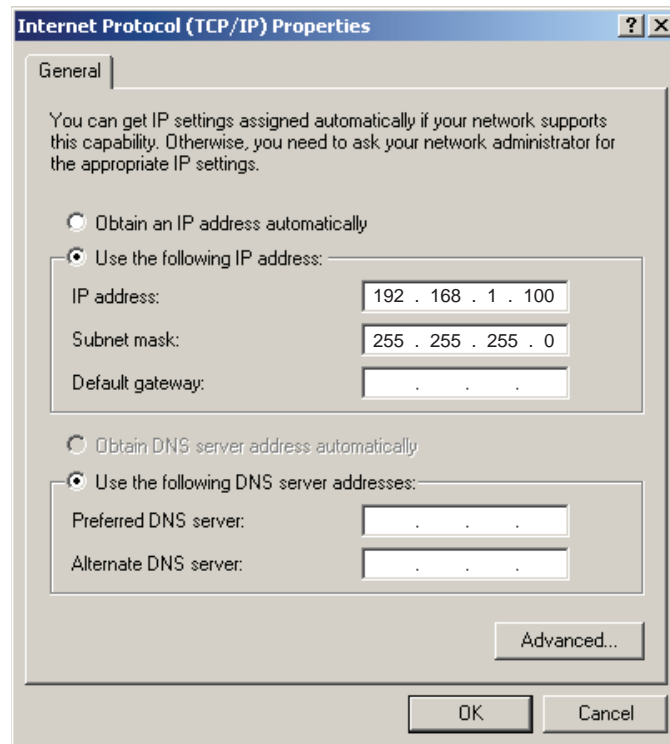
The following topics describe how to change the LAN network settings for computers that run Microsoft Windows operating systems.

Windows XP Internet Protocol (TCP/IP) Properties dialog box

You can review and change IP settings in the Internet Protocol Properties dialog box.

To open this dialog box in Windows XP:

1. Click **Start** and select **Settings**, then **Control Panel**.
2. Double-click **Network Connections**.
3. Double-click **Local Area Connection** and click **Properties**. The Local Area Connection Properties dialog box is displayed.
4. In the items list, double-click **Internet Protocol (TCP/IP)**. The Internet Protocol (TCP/IP) Properties dialog box is displayed.

Figure 39: Internet protocol (TCP/IP) Properties dialog box

5. Click **OK**.

Windows 2000 Internet Protocol (TCP/IP) Properties dialog box

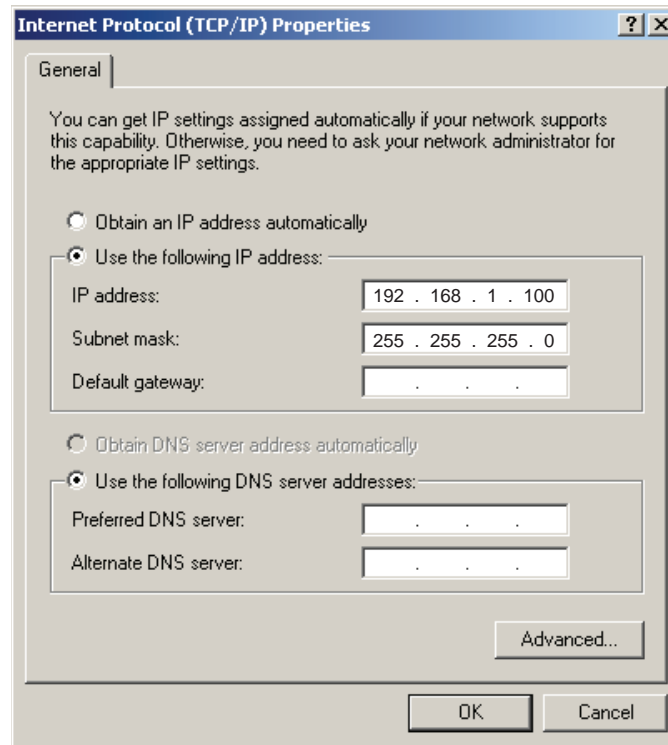
You can review and change IP settings in the Internet Protocol Properties dialog box.

To open this dialog box in Windows 2000:

1. Click **Start** and select **Settings**, then **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. In the Items list, double-click **Internet Protocol (TCP/IP)**. The Internet Protocol (TCP/IP) Properties dialog box is displayed.

Figure 40: Internet protocol (TCP/IP) Properties dialog box

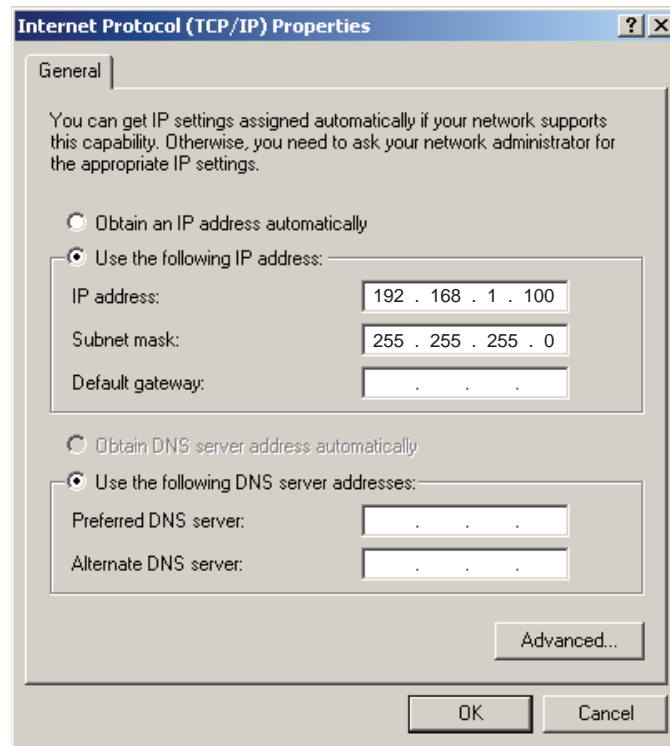


Windows Vista Internet Protocol Properties dialog box

You can review and change IP settings in the Internet Protocol Properties dialog box.

To open this dialog box in Windows Vista:

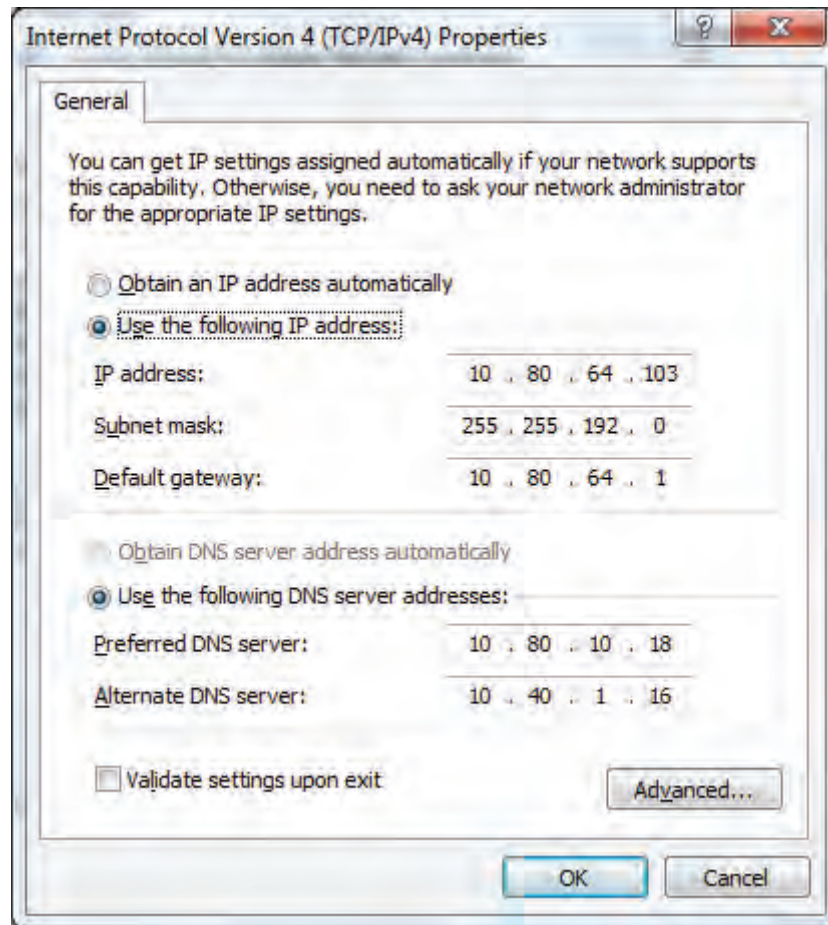
1. Click **Start** and select **Control Panel**.
2. Open **Network & Sharing Center**.
3. In the list, click **View Status** next to Connection. The Wireless Network Connection Status dialog box is displayed.
4. Click **Properties**. A permissions message is displayed.
5. If you are logged in as an administrator, click **Continue**. If you are not logged in as an administrator, enter the administrator's password to continue.
6. The Network Connection Properties dialog box is displayed.
7. Double-click **Internet Protocol Version 4 (TCP/IPv4)** in the items list. The Internet Protocol Properties dialog box is displayed.

Figure 41: Internet protocol (TCP/IP) Properties dialog box**Windows 7 Internet Protocol Version 6 (TCP/IPv6) Properties dialog box**

You can review and change IP settings in the Internet Protocol Version 4 (TCP/IPv4) dialog box.

To open this dialog box in Windows 7:

1. Click **Start** and select **Control Panel**.
2. Click **Network and Internet**.
3. Click **Network and Sharing Center**.
4. Click **Change Adapter Settings**.
5. On the connection you are using (usually Local Area Connection), right-click and select **Properties**. The Connection Properties dialog box is displayed.
6. 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 42: Windows 7 Internet Protocol Version 6 dialog box

Configure LAN settings through the front panel

All instruments need to be configured before they can be used on a network. The main parameters that need to be configured for IPv4 are:

- IP Addressing mode: Models 707B and 708B have a choice of Auto or Manual. Auto mode means the instrument will try to get an address through a DHCP server first and if this fails it will revert to Auto-IP mode. Manual is static IP addressing.
- Subnet Mask.
- Gateway.
- DNS Server address if you are using DNS.

Instruments with front panel displays and menus allow you to configure the instrument LAN settings through the front panel.

Web connection

You can enable or disable access to the instrument's web interface.

To enable or disable a web connection:

1. From the front panel, press the **MENU** key, and then select **LAN > ENABLE > WEB**.
2. Select either **ON** or **OFF**. After the power cycle reminder, you return to the LAN CONFIG menu.
3. Press the **EXIT (LOCAL)** key to return to the LAN MENU.
4. Turn the instrument on and off again to finalize the changes.

Telnet connection

Telnet is similar to raw socket and is used when the user needs to interact directly with the instrument, typically for debugging and troubleshooting. Telnet requires a separate telnet program.

The instrument supports the telnet protocol, which you can use over a TCP/IP connection to issue commands to the instrument. You can use a telnet connection to interact with scripts or issue commands in real time.

To enable or disable a telnet connection:

1. From the front panel, press the **MENU** key, and then select **LAN > ENABLE > TELNET**.
2. Select either **ON** or **OFF**. After the power cycle reminder, you return to the LAN CONFIG menu.
3. Press the **EXIT (LOCAL)** key to return to the LAN MENU.
4. Turn the instrument on and off again to finalize the changes.

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 VISA software is required. Test Script Builder (TSB) uses VISA and can be used with the VXI-11 interface. You can expect a slower connection with this protocol compared to a raw socket.

To enable or disable a VXI-11 connection:

1. From the front panel, press the **MENU** key, and then select **LAN > ENABLE > VXI11**.
2. Select either **ON** or **OFF**. After the power cycle reminder, you return to the LAN CONFIG menu.
3. Press the **EXIT (LOCAL)** key to return to the LAN MENU.
4. Turn the instrument on and off again to finalize the changes.

Raw socket connection

Raw socket is a basic Ethernet connection that communicates in a manner similar to RS-232. The instrument will always terminate messages with a line feed, but because binary data may include bytes that resemble line feed characters, it may be difficult to distinguish between data and line feed characters.

Use raw socket as an alternative to VXI-11. Raw socket offers a faster connection than VXI-11. However, raw socket does not support explicit message boundaries, serial poll, and service requests.

To enable or disable a raw socket connection:

1. From the front panel, press the **MENU** key, and then select **LAN > ENABLE > RAW**.
2. Select either **ON** or **OFF**. After the power cycle reminder, you return to the LAN CONFIG menu.
3. Press the **EXIT (LOCAL)** key to return to the LAN MENU.
4. Turn the instrument on and off again to finalize the changes.

Check the LAN network settings

You can check the network settings for the instrument without making changes.

To check the network settings:

1. From the instrument front panel, select **MENU > LAN > STATUS**.
2. Use the navigation wheel to select the following network settings:
 - **IP_ADDRESS:** The IP address that the instrument is using to communicate over the LAN.
 - **GATEWAY:** The gateway address that the instrument is using to communicate over the LAN.
 - **SUBNET:** The subnet mask that the instrument is using to communicate over the LAN.
 - **METHOD:** Automatic or Manual. When this is automatic, the instrument assigns LAN settings automatically. When this is manual, you need to set the LAN settings.
 - **DNS:** The DNS information.
 - **MAC-ADDRESS:** The Media Access Control address of the network interface card.
 - **SPEED:** The instrument automatically detects the speed of the LAN and adjusts its own settings to match.
 - **DUPLEX:** The instrument automatically detects the duplex setting of the LAN and adjusts its own settings to match.
 - **Port:** RAW-SOCKET, TELNET, VXI-11, or DST. Select the port type to see the assigned port number.
 - **Password:** The current password.
3. Press the **ENTER** key to view the setting.
4. Press the **EXIT** key once to return to the STATUS menu.

Set up automatic IP address selection

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

If you select AUTO, the instrument attempts to get an IP address from a DHCP server. If this fails, it reverts to an Auto-IP address.

NOTE

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

To set up automatic IP address selection:

1. On the host computer, in the Internet Protocol Properties dialog box, select **Obtain an IP address automatically**. This enables DHCP mode. If this fails, the computer will automatically try Auto-IP addressing (DLLA).
2. Click **OK**. Close the network settings dialog boxes.
3. From the front panel of the instrument, press the **MENU** key.
4. Use the navigation wheel to select **LAN > CONFIG > METHOD**. Press the navigation wheel to display the CONFIG menu.
5. Select **AUTO**. Press the navigation wheel to select AUTO and return to the LAN CONFIG menu.
6. Press the **EXIT (LOCAL)** key once to return to the LAN MENU.
7. Select **APPLY**. Press the navigation wheel to apply the setting. The Main Menu is displayed.

Set up instrument for manual LAN configuration

After setting up your computer for connection to the instrument, you configure the instrument's LAN settings through the instrument front panel. Settings include the IP address, subnet mask, and the default gateway.

To set up the LAN on the instrument:

1. From the front panel, press the **MENU** key.
2. Use the navigation wheel to select **LAN > CONFIG > METHOD**. Press the navigation wheel to display the METHOD menu.
3. Select **MANUAL**. Press the navigation wheel to select your choice and return to the LAN CONFIG menu.
4. Select **IP_ADDRESS**.
5. If you are:
 - Connecting directly to a host computer, refer to the recorded computer's IP address. Set the IP address to match the IP address of the host computer. Change the last three numbers (after the last decimal point) to a number that is unique on the LAN. The last three digits may be anything from 1 to 254 for a subnet mask of 255.255.255.0.
 - Connecting to a network: Enter the address provided by your system administrator.
6. Record the instrument's IP address.
7. Press the **ENTER** key when the IP address is complete. The LAN CONFIG menu is displayed.

8. Select **GATEWAY**.
9. Set the gateway value to match the gateway of the host computer.
10. Press the **ENTER** key when the gateway is complete. The LAN CONFIG menu is displayed.
11. Select **SUBNET**.
12. Set the subnet value to match the settings of the host computer or use the value supplied by your system administrator.
13. Press the **ENTER** key. The LAN CONFIG menu is displayed again.
14. Press the **EXIT (LOCAL)** key once to return to the LAN menu.
15. Select **APPLY**. Press the navigation wheel to apply the change. The Main Menu is displayed.

Change DNS settings

On the instrument, you can enable or disable the DNS settings and assign a host name to the DNS server.

When verify is enabled, the instrument performs a DNS lookup to verify that the DNS host name matches the value specified in the DNS server.

When dynamic is enabled, DNS registration works with the DHCP server to register the host name specified with the DNS server.

You can also define additional DNS addresses.

To change DNS settings:

1. From the front panel, press the **MENU** key, and then select **LAN > CONFIG > DNS > VERIFY**.
2. Select either **ENABLE** or **DISABLE**. You return to the DNS menu.
3. Select **DYNAMIC**.
4. Select either **ENABLE** or **DISABLE**. You return to the DNS menu.
5. To set an additional DNS addresses, select **DNS-ADDRESS1** or **DNS-ADDRESS2**.
6. Enter the address.
7. Press the **ENTER** key.
8. Press the **EXIT** key twice to return to the LAN MENU.
9. Select **APPLY**.

Change the IP configuration through the web interface

The LAN settings, such as IP address, subnet mask, gateway, and DNS address, can be changed through the web page of the instrument.

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

NOTE

You can also change the IP configuration using ICL commands. See [lan functions and attributes](#) (on page 5-12) for more information.

To change the IP configuration using the instrument web page:

1. Access the instrument's internal web page (see [Using the web interface](#) (on page 2-69)).
2. From the navigation bar on the left, in the LXI Home menu, select **IP Config**.
3. Click **Modify**.
4. You are prompted for a password. The default is **admin**.

Figure 43: Modify IP Configuration page

Hostname:	<input type="text" value="K-708B-01234567"/>
Description:	<input type="text" value="Keithley 708B #01234567"/>
TCP/IP Configuration Mode:	<input type="radio"/> Automatic <input checked="" type="radio"/> Manual
Static IP Address:	<input type="text" value="10.60.8.83"/>
Subnet Mask:	<input type="text" value="255.255.255.0"/>
Default Gateway:	<input type="text" value="10.60.8.1"/>
DNS Servers:	<input type="text" value="10.80.10.18"/> <input type="text" value="10.40.1.16"/>
Domain:	<input type="text"/>
Dynamic DNS:	<input type="radio"/> Enabled <input checked="" type="radio"/> Disabled

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

NOTE

You may lose your connection with the embedded web interface after clicking **Submit**. This is normal and does not indicate an error or failure of the operation. If this occurs, re-open the instrument's web page to continue.

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

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) (<http://www.keithley.com/support>).

VISA

The Virtual Instrument Software Architecture (VISA) is a standards body that maintains the specifications for a whole series of software components related to instrument connectivity (I/O). The VISA specifications, formerly maintained by the VXIplug&play Systems Alliance, are now being maintained by the [IVI Foundation](http://www.ivifoundation.org) (<http://www.ivifoundation.org>).

The VISA library (standard VPP-4.3) is a standard for an API to communicate with instruments connected to the computer communication buses: Ethernet, USB, RS-232, GPIB, and so on. VPP-4.3 is a software API standard — several instrument vendors have implementations of VPP-4.3, including National Instruments, Agilent Technologies, and Tektronix.

There are three types of programming interfaces to VISA: VISA-C, VISA-COM and VISA .NET.

- VISA-C is a DLL that has a flat API. The five main message-based functions are viOpenDFLT, viOpen, viWrite, viRead and viClose.
- VISA-COM is an ActiveX type interface to VISA that is more suited for the VB6 and .NET environments.
- VISA.NET is a .NET interface for VISA that the IVI Foundation is currently (July 2010) standardizing on.

VISA shared components

The IVI Foundation provides the VISA shared components, which contain the VISA-COM components that VISA vendors install with their VISA installations. There are separate entries in the Add/Remove Programs dialog box for the VISA shared components and the vendor's VISA.

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.

NI CVI runtime engine

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

NI IVI Compliance Package

The 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 SCPI-based instrument driver

The Keithley SCPI-based Instrument IVI-C Driver is used to support the Keithley Configuration Panel Wizard and Keithley Communicator functionality. It contains simple functions for opening, configuring, taking measurements from, and closing the instrument.

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 TSB.

The KIOL contains:

- NI VISA Runtime
- Keithley Configuration Panel
- Keithley Communicator

NI VISA Runtime

NI VISA is National Instruments 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 runtime-only files. The runtime version contains only the binary files (DLLs) that allow the drivers to operate.

The 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 VISA runtime 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 auto-detect 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:

- Windows 7 (32-bit only)
- Windows Vista (SP1)
- Windows XP (SP2)
- Windows 2000 (SP4)

Note that Windows 95, Windows ME, Windows 98, and Windows NT 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 Runtime x.x.x (If present) (x.x.x is the VISA version)
3. Reboot your computer.

How to install the Keithley I/O Layer

NOTE

Before installing, it is a good idea to check to see if a later version of the Keithley I/O Layer is available. Check the [Keithley Instruments website](http://www.keithley.com) (<http://www.keithley.com>). Select the Support tab, type in KIOL, and select software driver.

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

The software installs the following components:

- Microsoft .NET Framework 2.0
- NI IVI Compliance Package 3.3
- NI-VISA Runtime 4.5.0
- Keithley SCPI-based Instrument IVI-C driver SCPI-856C02
- Keithley I/O Layer KIOL-850C02

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

1. Close all programs.
2. Place the CD into your CD-ROM drive.
3. Your web browser should start automatically and display a screen with software installation links. To manually open the web page, use a file explorer to navigate to the CD drive and open the file named `index.html`.
4. From the web page, select the Software category and then click on the Keithley I/O Layer.
5. Accept all defaults. Click **Next**.
6. Click **Install**.
7. 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) (<http://www.keithley.com>). 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. 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 Instrument 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 vendor 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, it is recommended that you do not reinstall it as 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 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 2-62).
2. Rerun the KIOL installer. Note where the installer unpacks the files (usually in a temporary folder).
3. Cancel the installer.
4. Go to 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)
 - Visa Runtime
 - KIOL
 - Keithley SCPI Driver
6. Ignore all the other folders.
7. Reboot the computer.

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

The Keithley I/O Layer chains many other installers together.

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

- National Instruments NI IVI Compliance Package
- National Instruments NI-Visa Runtime
- 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 on the format of the resource string, refer to the VISA specification VPP4.3 at the IVI Foundation web site or refer to the help file 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. Models 707B and 708B support 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 Models 707B and 708B, the port is 5025. See [Instrument LAN protocols](#) (on page 2-40) 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 hex code for the instrument vendor company. Keithley Instruments is 05E6.

model code is the model number of the instrument. For example, when addressing a Model 707B, use 707B.

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](#) (on page 2-31).

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 is more than one in the computer. If there is only one GPIB card, skip *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 need be.

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;
    }
}
```

Using the web interface

Introduction

The Model 707B or 708B web interface allows you to 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 2-81) page of 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 communications](#) (on page 2-36) and follow the instructions in [LAN quick start](#) (on page 2-36).

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 instrument web interface gives you basic information about the instrument, including:

- The model, serial number, firmware revision, raw socket, and LXI information
- A list of slots and the switch cards installed in each slot
- An ID button to help you locate the instrument

Identify the instrument

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

Make sure you have a remote connection to 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.

Figure 44: ID lit



Click **ID** again to return the button to its original color and return the LAN status indicator to steady.

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:

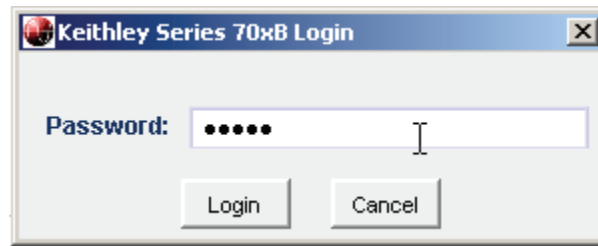
1. Open a page that contains a **Login** button, such as one of the Cards pages, Scan Builder, or TSB Embedded.
2. Click **Login**. The login dialog box is displayed.

Figure 45: Log in



3. Enter the password (the default is **admin**).

Figure 46: Enter 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**.

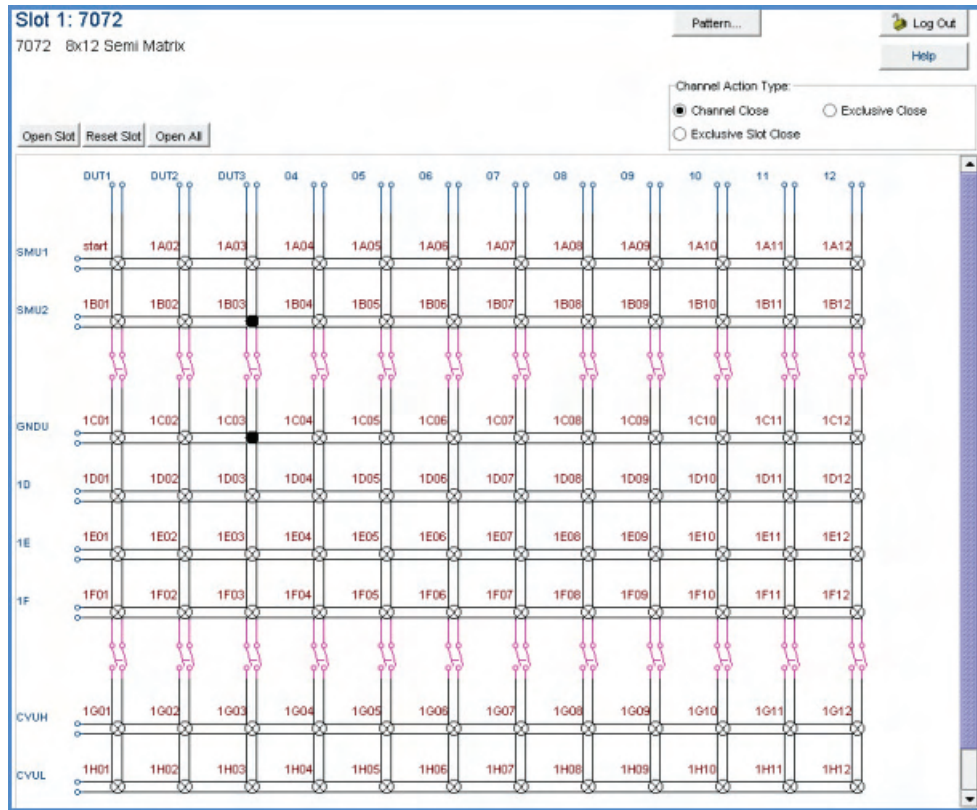
Card pages

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. This page displays a grid that shows the relay configuration of the switch card.

Figure 47: Web interface Cards page



Open and close slots from the card pages

You can open and close slots 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 is a circle with an x in it:

Figure 48: Web interface open channel



When the channel is closed, the connection is black:

Figure 49: Web interface closed channel



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**.

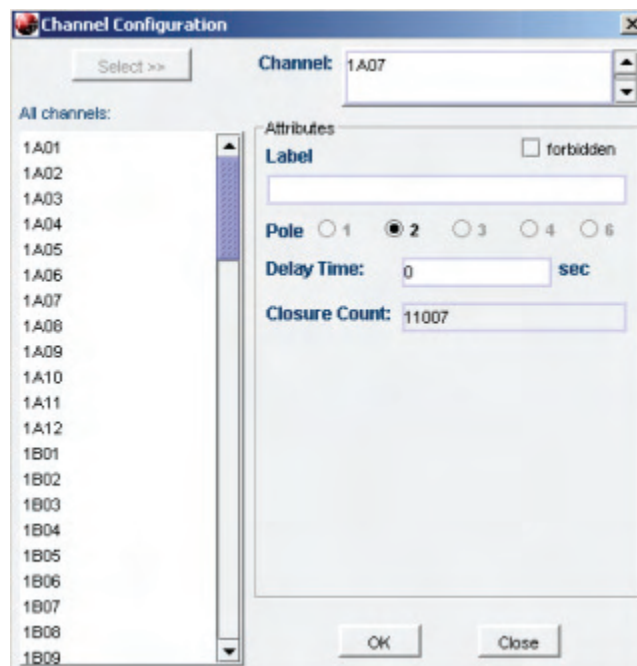
You can open all channels in the instrument by clicking **Open All**.

For more information on opening and closing channels, see [Closing and opening channels](#) (on page 2-94).

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 50: 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. **Delay Time:** The additional delay to incur after the relay settles when closing. Enter the value for the delay in seconds. The total delay for channel close is this delay plus the relay settling time.

This dialog box also displays the closure count. See [Determining the number of relay closures](#) (on page 2-93) for information.

Set up channel patterns from the web interface

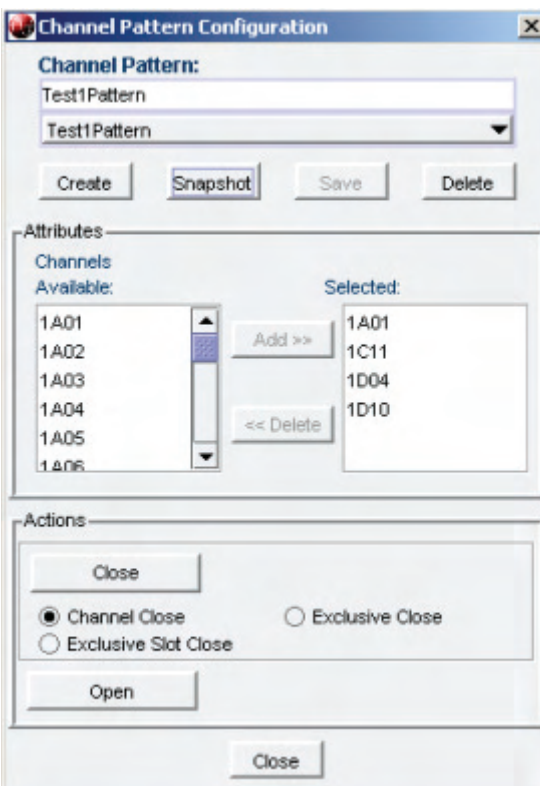
You can use channel patterns as a convenient way to refer to a group of switching channels with a single alphanumeric name. When you perform close or open operations on a channel pattern, only the channels that are in the channel pattern are affected.

There is no speed advantage in 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 51: 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 2-99).

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 open
- Labels return to default of slot, row, column
- Delays are set to zero
- If the channel is forbidden to close, it is cleared from being forbidden to close
- 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 3-1).

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 Model 707B or 708B 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 2-87) 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.
3. If you selected Pattern, select a pattern from the Channel Pattern list.
4. If you selected Number, select the channel numbers from the list. You can use Ctrl+click to select multiple channels and Shift+click to select a range of channels. To remove your selections from the Add Channel By list, click **Clear Channel Selection**.
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. 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 check the Query State box 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 Triggers tab. You can set:

- The event that starts the scan
- The time interval between scans
- The time interval between channels

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

Trigger to start scan

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 to start the scan. This is the default selection.

Digital Input: When selected, you select a 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 [Advanced triggering](#) (on page 2-80) for other options.

Channel action trigger

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.

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 [Advanced triggering](#) (on page 2-80) for other options.

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.

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](#) (on page 3-1) 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 3-1).

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.
- **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).
- **Channel Action Stimulus:** Select the event that causes the channel event detector to be set to the detected state (the step can begin).
- **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.

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.

To set the scan mode from the web interface:

1. Select the **Advanced Trigger** tab.
2. Select **Mode**.
3. Select **OPEN_ALL** or **OPEN_SELECT**.

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.

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 create a script, see [Create a script using TSB Embedded](#) (on page 2-82).

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

To export the selected script to the computer, click **Export to PC**. Choose the directory where you want to save the script and click **Save**. 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 to 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**.

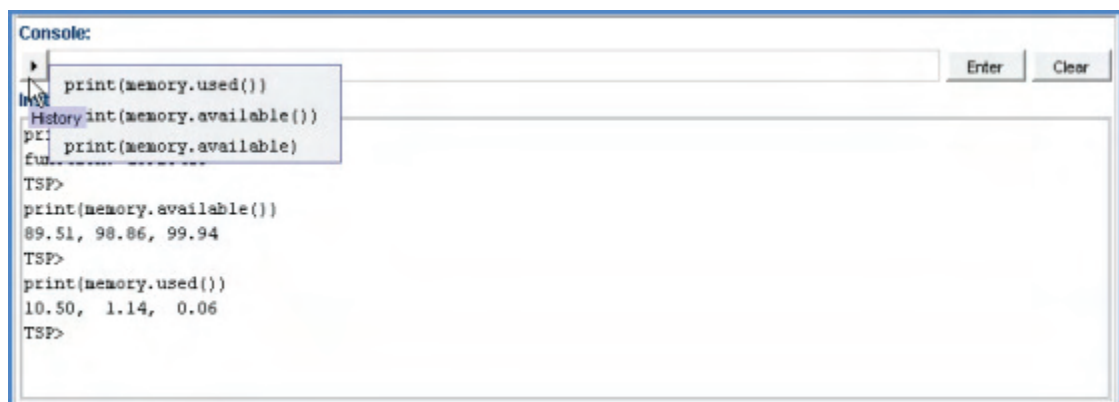
You can type the name of a script in the **TSP Script** box and click **View Script** to view the contents of the script.

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.

Figure 52: Model 707B or 708B web interface console



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`. For information on using TSB Embedded, select the Help button on a web page or the Help option from the navigational pane on the left hand side.

You can create a script in from the instrument web page with TSB Embedded. When you save the script in TSB Embedded, it is loaded into the runtime environment and saved in the nonvolatile memory of the instrument.

To create a script using TSB Embedded:

1. In the TSP Script box, enter the name of the script.
2. In the input area, enter the sequence of commands to be included in the script. Line numbers are automatically assigned.
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

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 and number of poles.
- Review the number of closures for each channel on the selected slots. The counts reported for the following cards indicate the number of closures since the last power cycle of the card:
 - 7072
 - 7072-HV
 - 7173-50
 - 7174A
- For all other cards, 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 Model 707B or 708B is an LXI Class C 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.

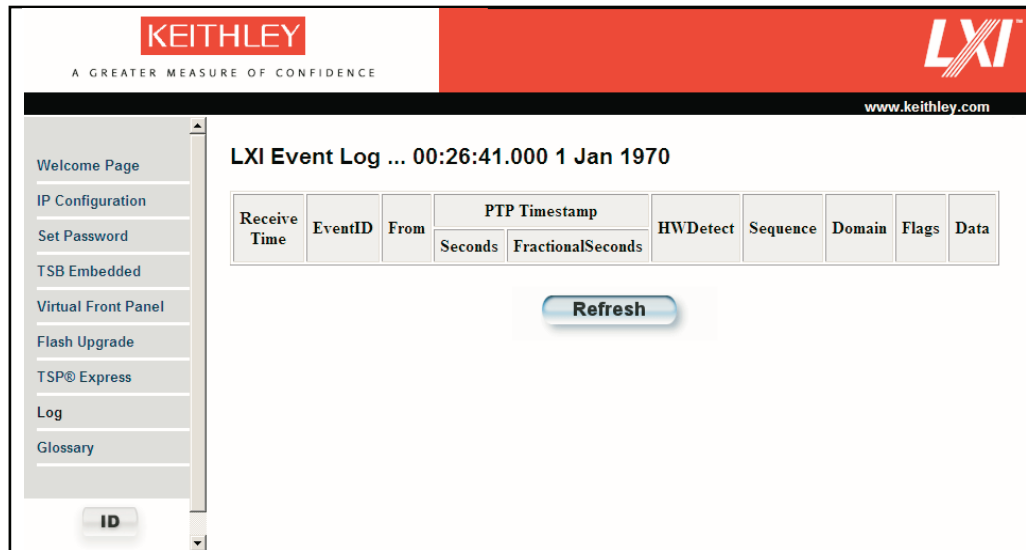
See [Change the IP configuration through the web interface](#) (on page 2-56) for more information.

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 53: Event log



The time stamp, 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 Model 707B or 708B 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 Model 707B or 708B does not support the IEEE-1588 standard; the values in this field are always 0 (zero)	

Switch operation

Working with channels

Overview

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 the Models 707B and 708B Quick Start Guide.

The Models 707B and 708B have specific settings that apply to opening and closing channels.



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.

Channel identification

The channels on the cards that you can use with the Model 707B or 708B are referred to by a channel specifier. The specifier is used to identify channels for use with close and open operations, scans, and channel patterns, using the front panel, web, or remote command interface.

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

Channel types

The Models 707B and 708B support matrix cards. The documentation for your specific card model lists the available channels.

Specify multiple channels using lists. Lists build on the individual channel specifier.

Channel identifiers

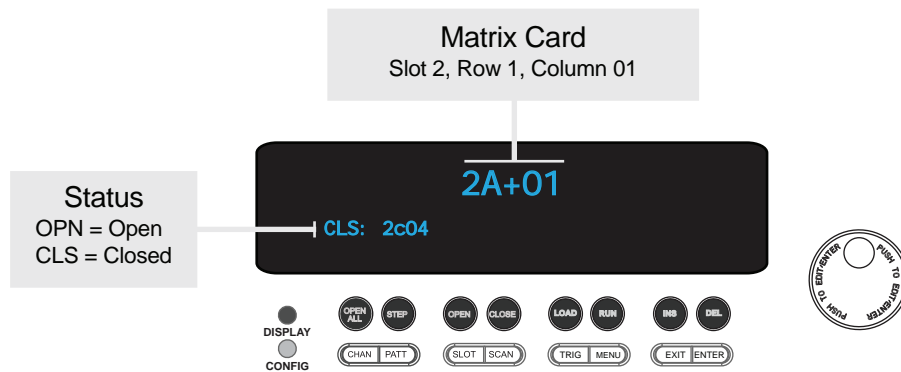
The channels on the matrix cards are referred to by their slot, bank, row, and column numbers:

- **Slot number:** The number of the slot in which the card is installed.
- **Bank:** 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 counting sequence is: 98, 99, A0, A1, ..., A8, A9, B0, B1, ...

Matrix channel examples

Reference	Slot	Bank	Row	Column
1A05	1	N/A	1	05
1C05	1	N/A	3	05
3C12	3	N/A	3	12

Figure 54: Matrix card display showing channel identifier



Channel list parameter for remote commands

The channel list parameter is a string-type parameter that is used when controlling the relays of the Model 707B or 708B 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 2-99).
- The string may contain a single channel, single channel pattern, multiple channels, or multiple channel patterns that are 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 row 1 and columns 3 and 5 of slot 1, use ("1A03, 1A05") for *channelList*.
- To perform an open or close operation on all channels in the range of row 1 and columns 1 through 5 of slot 1, use ("1A01:1A05") for *channelList*.

Queries that return a list of channels

For queries that return a channel list parameter, the Model 707B or 708B separates the channels by a comma or semicolon, depending on the command. When multiple channels are used in the query, the information for the lowest numbered channel is listed first, increasing to the highest numbered channel.

When multiple slots are used in the query, the information for the lowest slot number is listed first and increases to the highest slot and channel.

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 [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) 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.

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 7-21)

Determining the number of relay closures

The Models 707B and 708B 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.

NOTE

The counts reported for the following cards indicate the number of closures since the last power cycle of the card:

7072


7072-HV

7173-50

7174A

For all other cards, 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:

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()`.

Identifying installed switching cards

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 card 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 for Model 707B or 1 for Model 708B)

Example

To get a list of all switching cards installed in the slots of a Model 707B, 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:

```
7174, 8x12 Fast Low-I Matrix, 01.00a, <Module Serial Number>
7072, 8x12 Semi Matrix, 01.00a, <Module Serial Number>
Empty Slot
Empty Slot
Empty Slot
Empty Slot
```

Selecting a channel from the front panel

You can perform operations on a single channel from the front panel. To select a channel, see [Selecting channels from the front panel](#) (on page 2-17). Once a channel is selected, it is the selected channel for front panel operation.

Closing and opening channels

Switching channels allow various methods for closing and opening relay channels.

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 is the selected channel).
- Channel exclusive slot close: Close the selected channel and open any closed channels in the same slot.
- Channel open.

The Model 707B or 708B verifies that the operation being requested for a channel is supported by the specified channel and that the channel exists in the instrument.

NOTE

You can use scans to perform a user-specified sequence of close and open operations on multiple channels. Refer to [Scanning and triggering](#) (on page 3-1) for information on scan operations.

Close and open channel operations and commands**NOTE**


When you turn on the Model 707B or 708B, relays for all switch cards in the instrument are opened.

When you close channels, you have the option to do an exclusive close for the channel or slot. An exclusive channel close closes the selected channel and opens any other channels that are closed.

You can also choose an exclusive slot close, which closes the selected channel and opens any other channels that are closed on the same slot as the selected channel. With an exclusive slot close, any channels that are closed on other slots remain closed.

You can use the front panel **CLOSE** and **OPEN** keys to open and close channels.

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

To close or open a channel from the web interface:

1. From the list on the left, select the slot that contains the channel you want to open or close.
2. If you are closing a channel, from the Channel Action Type list in the upper left, select the type of close.
3. Right-click the channel you want to open or close. If the channel is closed, it opens; if it is open, it closes.

To close or open a channel from the remote interface:

You can also open and close channels using the following commands:

- `channel.close()` (on page 7-19)
- `channel.exclusiveclose()` (on page 7-24)
- `channel.exclusiveslotclose()` (on page 7-25)
- `channel.open()` (on page 7-35)



NOTE

Refer to the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) for details on commands.

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:

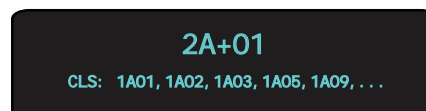
1. Go to channel view.
2. Select the channel you want to open or close.
3. Press **CHAN**.
4. Use the navigation wheel  to select the option. You can select:
 - **OPEN**: Opens the selected channel.
 - **CLOSE**: Closes the selected channel.
 - **EXCLOSE**: Closes the selected channel; opens any other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

Viewing the close or open status of a channel

To determine whether a channel 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.



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.

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 7-44)
```




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.

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 7-45)
- `channel.clearforbidden()` (on page 7-18)

Set up labels




You can define labels for rows, columns, and channels.

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

Channel labels can be up to 19 characters. Row and columns labels can be up to 8 characters. On the crosspoint display, the first four characters of the label are displayed. On the bottom display, the full label is displayed.

You can only set labels for slots and 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 the type of label you want to define:
 - **LABEL**: Sets the label that is displayed on the front panel for the specified crosspoint.
 - **LABEL-ROW**: Sets the label that is displayed on the front panel for the specified row.
 - **LABEL-COL**: Sets the label that is displayed on the front panel for the specified column.
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. Click **OK**.

To set up labels from the remote interface:

Use the commands:

- `channel.setlabel()` (on page 7-46)
- `channel.setlabelrow()` (on page 7-49)
- `channel.setlabelcolumn()` (on page 7-47)

You can use labels to refer to the channels in commands. For example, if you set the label for channel 1A01 to "start", you could use "start" to close and open the channel. If you set the row 1B to "SMU2" and the column 02 to "DUT2", you could close the channel by sending the command

```
channel.close("SMU2+DUT2")
```

This is shown in the following example:

```
channel.setlabel('1A01','start')
channel.close("start")
channel.setlabelrow("1B01", "SMU2")
channel.setlabelcolumn("1B02", "DUT2")
channel.close("SMU2+DUT2")
print(channel.getclose("allslots"))
```

Channel patterns

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

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

NOTE

Channel patterns inherit the delay times of the individual channels that comprise the pattern. For information on the sequence of close operations on multiple channels, refer to [Connection methods for close operations](#) (on page 2-89).

When you create a channel pattern, make sure to:

- Include all of the channels 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:

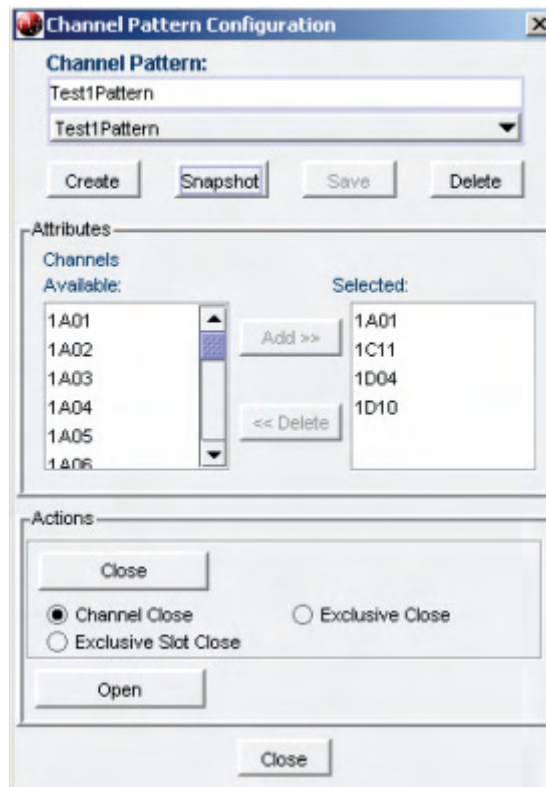
- The first character of the name must be alphabetic (upper or lower case)
- Names are case sensitive
- Pattern names must be different than row, column, and channel labels

To create a channel pattern from the front panel:

1. Close the channels you want to include in the channel pattern.
2. Press the **PATT** key.
3. From this menu, select the **CREATE** menu item.
4. From this menu, select the **SNAPSHOT** menu item.
5. At the prompt, enter a pattern name using the navigation wheel.
6. Use the **ENTER** key to apply the selection.
7. Use the **EXIT** key to leave the menu.

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 55: 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**.
6. When the Selected channel list is complete, click **Create**.

NOTE

If you close the channels you want to add to the new pattern, you can enter a name and click **Snapshot** to create the new pattern. The closed channels are added to a new pattern.

To create a channel pattern from the remote command interface:

You can also work with channel patterns using the following commands:

```
channel.pattern.catalog() (on page 7-37)
channel.pattern.delete() (on page 7-38)
channel.pattern.getimage() (on page 7-38)
channel.pattern.setimage() (on page 7-39)
channel.pattern.snapshot() (on page 7-41)
```

NOTE

Refer to the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) for more details.

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.

To perform a particular operation on a channel pattern, use the appropriate open or close command with the channel pattern name for the `channelList` parameter. Refer to [Close and open channel operations and commands](#) (on page 2-95) for detail.

When you request a close or open operation, the Model 707B or 708B 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.

To close or open the channels in a channel pattern from the front panel:

1. Press the **PATT** key to display a channel pattern (you might need to press **DISPLAY** first).
2. Select the channel pattern you want to open or close.

NOTE

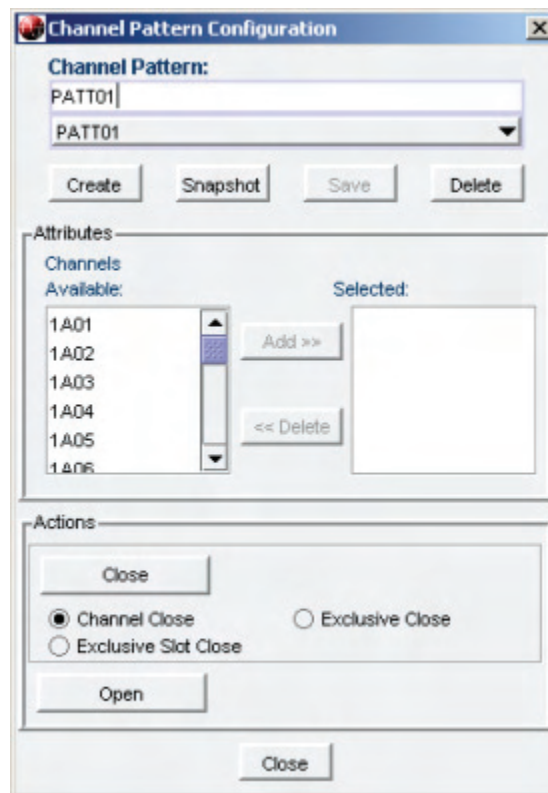
Model 707B only: Note that when you select a pattern, on the crosspoint display, the lights for the channels included in the pattern are dimly lit.

3. Perform any of the following actions:
 - Open the channels in the channel pattern: Press **OPEN**.
 - Close the channels in the channel pattern without affecting any other channels: Press **CLOSE**.
 - Close the channels in the channel pattern and open any other closed channels on the instrument: Select **PATT** and select EXCLOSE. Press **ENTER** to open or close the channels.
 - Close the channels in the channel pattern and open any other closed channels on the slot: Press **PATT** and select EXSLOTCLOSE. Press **ENTER** to open or close the channels.

To close or open the channels in a channel pattern from the web interface:

1. From the list on the left, select a slot with an installed card.
2. Click **Pattern**. The Channel Pattern Configuration dialog box is displayed.

Figure 56: Channel Pattern Configuration dialog box



3. Select the pattern.
4. In the Actions box, to:
 - Open the channels in the channel pattern: Click **OPEN**.
 - Close the channels in the channel pattern without affecting any other channels: Select Channel Close, then click **CLOSE**.
 - Close the channels in the channel pattern and open any other closed channels on the instrument: Select Exclusive Close and click **CLOSE**.
 - Close the channels in the channel pattern and open any other closed channels on the slot: Select Exclusive Slot Close and click **CLOSE**.

To close or open the channels in a channel pattern from the remote interface:

- `channel.close()` (on page 7-19)
- `channel.exclusiveclose()` (on page 7-24)
- `channel.exclusiveslotclose()` (on page 7-25)
- `channel.open()` (on page 7-35)

NOTE

Refer to the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) for detail on each command.

Channel pattern storage

Channel patterns are:

- Part of the script that is created with Create Config Script.
- Deleted when the instrument is reset.
- Deleted when a channel associated with the pattern is reset.
- Allocated 32KB of memory in the Models 707B and 708B instrument for all channel patterns.

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 7-135) or `memory.used()` (on page 7-137).

Reset a channel

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

- A closed channel opens
- Additional user delay is set to zero
- Labels return to default value
- If the channel is forbidden to close, it is cleared from being forbidden to close
- If the channel is used in channel patterns, the channel patterns that contain the channel are deleted.


Using this function to reset a channel 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 7-43).

Pseudocards

You can perform open, close, and scan operations and configure your instrument 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 (LOCAL)** key to take the instrument out of remote mode.

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

Pseudocards programming example

Use the following command to set the pseudocard of slot 2 for 7072 8x12 Semi Matrix card simulation:

```
slot[2].pseudocard = 7072
```

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 6-1). For more information on the autoexec script, see [Autoexec script](#) (on page 6-8).

A sample configuration script is shown in the following example.

<pre>--Auto created configuration script</pre>	<p>Indicates that this was created with the Create Configuration Script feature</p>
<pre>--Tue Jul 13 13:02:12 2010</pre>	<p>Date and time stamp</p>
<pre>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") else</pre>	<p>Code that verifies that card and slot are in agreement</p>
<pre> reset()</pre>	<p>Reset command</p>
<pre> 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")</pre>	<p>Code that captures the non-factory default settings</p>
<pre>end</pre>	

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.

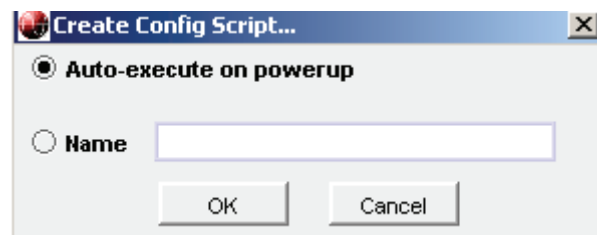
To create a configuration script from the front panel:

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 57: 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 6-6) for information.

Functions and features

In this section:

Scanning and triggering	3-1
Trigger model	3-1
Scan and step counts.....	3-4
Basic scan procedure.....	3-5
Remote interface scanning.....	3-8
Hardware trigger modes.....	3-12
Understanding synchronous triggering modes	3-17
Events	3-21

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 Model 707B or 708B 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.

Trigger model

When you run a scan, the scan sequence follows a trigger model. The trigger model is shown in the following flowchart.

NOTE

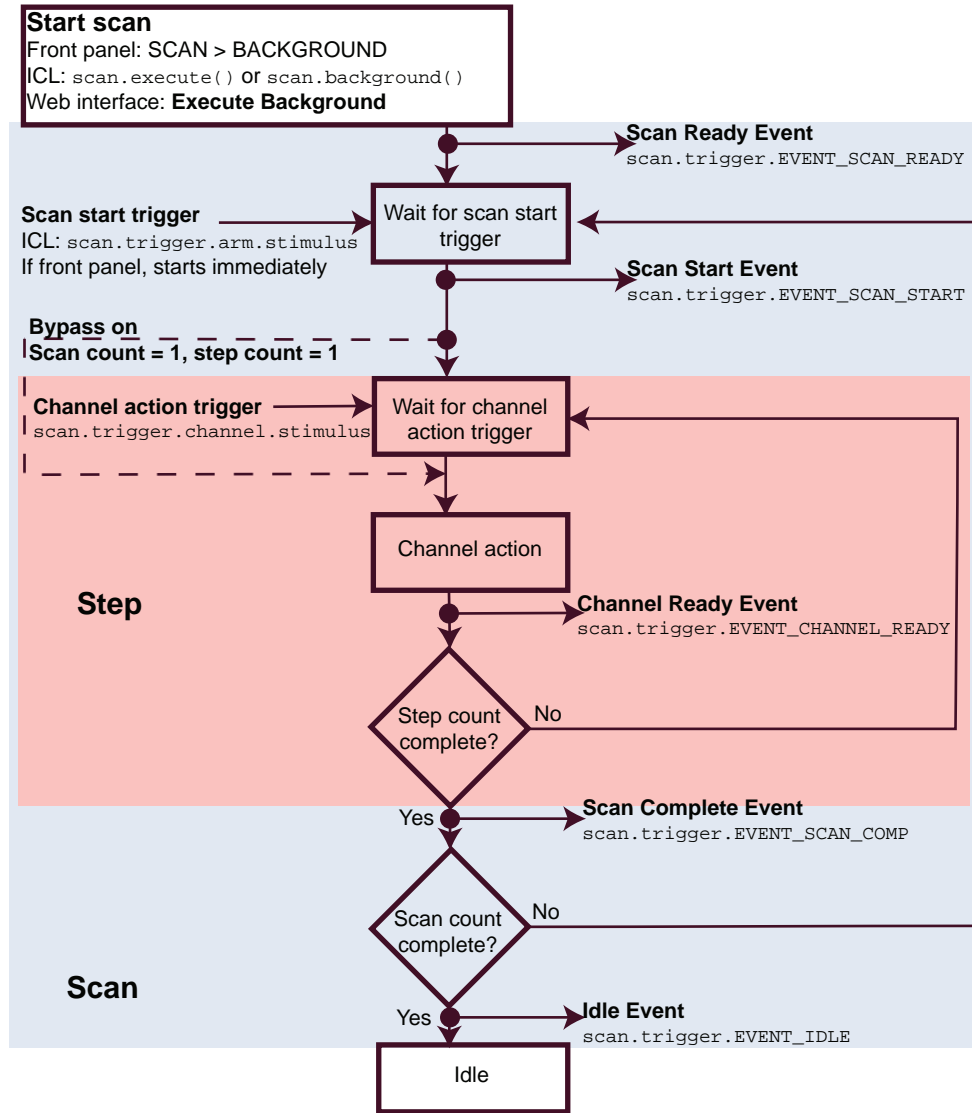
In Models 707B and 708B, only scanning operations use the trigger model. Individual open and close commands do not affect the trigger model.

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 58: Models 707B and 708B triggering overview



Trigger model components

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

Start scan

When a scan is initiated, the instrument leaves the idle state and prepares to start scanning. To prepare for scanning, it verifies that channel settings match the scan settings (such as opening all channels or opening only channels on specific slot). After preparation is complete, the instrument generates the Scan Ready Event and either starts immediately or waits for the arm stimulus event.

When the scan is complete, the instrument returns to the idle state.

Wait for scan start trigger

The scan can start immediately after receiving the Scan Ready Event, or the instrument can wait for a start trigger (also called the arm action). If it waits for a start trigger, the scan does not start until it receives the start trigger.

When the scan starts, the instrument generates the Scan Start Event.

Wait for channel action trigger

The channel action trigger is an event that tells the instrument to begin processing the next channel action.

You can bypass the channel action trigger if this is the first step of the first scan count. Bypass is available from the front-panel **CONFIG+SCAN** menu, or by using the command `scan.bypass`.

Channel action

During the channel action, the instrument opens or closes channels as needed.

When the channel actions are complete, the instrument generates the Channel Ready Event.

Step count complete

The trigger model repeats these actions for every step in the scan list. The instrument loops back to wait for channel action trigger for each step until the steps are complete.

When all the steps are complete, the instrument generates the Scan Complete Event.

Scan count complete

The scan can be set to repeat. If the scan count is not complete, the instrument loops back to wait for the scan start trigger.

Idle

After the instrument has completed the scan list the requested number of times, the instrument generates the Idle Event. This is the completion of the scan.

Trigger model events and associated commands

The Models 707B and 708B 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 ICL attribute
Scan Ready Event	<code>scan.trigger.EVENT_SCAN_READY</code>
Scan Start Event	<code>scan.trigger.EVENT_SCAN_START</code>
Channel Ready Event	<code>scan.trigger.EVENT_CHANNEL_READY</code>
Scan Complete Event	<code>scan.trigger.EVENT_SCAN_COMP</code>
Idle Event	<code>scan.trigger.EVENT_IDLE</code>

NOTE

Scanning operations run through the trigger model, but individual open and close 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.

Basic scan procedure

NOTE

It is always better to configure all channel attributes before creating a scan.

To perform a scan:

1. Configure the channels for scanning as needed.
2. Build the scan list:
 - **Front panel:** Press the **INS** key. The steps are executed in the order in which they are added.
 - **Remote interface:** Send the `scan.create()`, `scan.add()`, or `scan.addimagestep()` command.
3. Configure the scan settings (for example, scan count, bypass, mode, and so on).
4. To start the scan:
 - **Front panel:** Press the **STEP** key or the **SCAN** key and select the **BACKGROUND** menu item.
 - **Remote interface:** Send the command `scan.execute` or `scan.background`.
5. The trigger model leaves the idle state and performs actions on the channels involved in scanning.
 - **Front panel:** When you press the **STEP** key, the Models 707B and 708B leave the idle state and perform the channel action associated with the first step in the scan list.

NOTE

While scanning is enabled, pressing most front panel keys will display the message "ERROR CODE: 5522 Scan Running, Must Abort Scan."

6. The channels are scanned or stepped in the order they were added to the list.
 - **Front panel:** If you are stepping through the scan, press the **STEP** key to proceed to the next step in the list.
 - **Remote interface:** You cannot step a scan remotely over the bus.
7. 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, channel states match the aborted state of channels in terms of which are closed and opened.

If configured to scan the channels in the scan list again, the Model 707B or 708B waits at the control source for another trigger event. After the scan is complete, the Model 707B or 708B 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.

Changing attributes of an existing scan

When a scan already exists, changing channel attributes also causes the scan to change. Once a scan list has been defined, the Model 707B or 708B tries to incorporate your changes into the scan. 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. On the front panel, press the **SCAN** key and select the **LIST** option, using the navigation wheel to scroll through the options. For remote operation, use the `scan.list()` function.

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.

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 INS** key. The related ICL is `scan.add`.
 - **BYPASS**: Enables (ON) or disables (OFF) bypassing the first step of the first scan pass. Related ICL command: `scan.bypass` (on page 7-145).
 - **MODE**: Sets the scan mode value to one of the following:
 - OPEN_ALL (default setting)
 - OPEN_SELECT
 - Related ICL command: `scan.mode()`.
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 INS 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 **INS** key to add the selected channels or pattern to the existing scan list.

Press the **DEL** key to remove the selected channels or pattern from the existing scan list. Only the first occurrence of the selected item is removed.

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 **DEL** key.

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. If a scan is running in the foreground, it will need to finish or be aborted before you can query any settings.

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(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:

- [trigger.blender\[N\].clear\(\)](#) (on page 7-201)
- [trigger.blender\[N\].orenable](#) (on page 7-202)
- [trigger.blender\[N\].overrun](#) (on page 7-203)
- [digio.trigger\[N\].clear\(\)](#) (on page 7-57)
- [digio.trigger\[N\].pulsewidth](#) (on page 7-60)
- [digio.trigger\[N\].stimulus](#) (on page 7-62)
- [digio.trigger\[N\].wait\(\)](#) (on page 7-64)
- [lan.trigger\[N\].assert\(\)](#) (on page 7-114)
- [lan.trigger\[N\].clear\(\)](#) (on page 7-115)
- [lan.trigger\[N\].overrun](#) (on page 7-121)
- [lan.trigger\[N\].stimulus](#) (on page 7-123)
- [lan.trigger\[N\].wait\(\)](#) (on page 7-125)
- [scan.add\(\)](#) (on page 7-142)
- [scan.background\(\)](#) (on page 7-144)
- [scan.bypass](#) (on page 7-145)
- [scan.create\(\)](#) (on page 7-146)
- [scan.execute\(\)](#) (on page 7-147)
- [scan.list\(\)](#) (on page 7-148)
- [scan.mode](#) (on page 7-150)
- [scan.reset\(\)](#) (on page 7-150)
- [scan.scancount](#) (on page 7-151)
- [scan.state\(\)](#) (on page 7-152)
- [scan.stepcount](#) (on page 7-153)
- [scan.trigger.arm.clear\(\)](#) (on page 7-153)
- [scan.trigger.arm.set\(\)](#) (on page 7-154)
- [scan.trigger.arm.stimulus](#) (on page 7-154)
- [scan.trigger.channel.clear\(\)](#) (on page 7-156)

- [scan.trigger.channel.set\(\)](#) (on page 7-156)
- [scan.trigger.channel.stimulus](#) (on page 7-157)
- [scan.trigger.clear\(\)](#) (on page 7-158)

Scanning examples

Assume you have a 7072 card installed in slot 1 of your instrument and you want to scan column 5 on rows 1 to 3. To create this scan, send the following command:

```
scan.create("1A05, 1B05, 1C05")
```

To see the scan list generated from this command, send:

```
print(scan.list())
```

The following will be output:

```
Init) OPEN...
 1) STEP: 1A05
    CLOSE: 1A05
 2) STEP: 1B05
    OPEN: 1A05
    CLOSE: 1B05
 3) STEP: 1C05
    OPEN: 1B05
    CLOSE: 1C05
```

This indicates that the scan list includes three steps. For step 1, the running scan will close "1A05". For step 2, it opens "1A05" and closes "1B05". For step 3, it will open "1B05" and close "1C05".

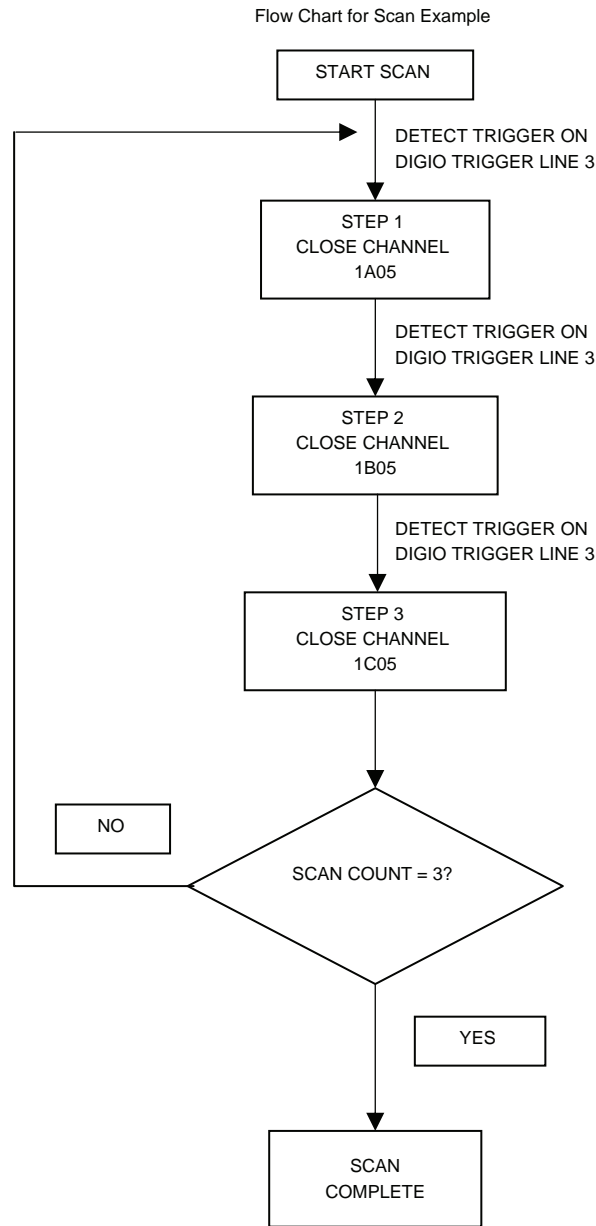
The following table illustrates how scan count works with this scan list to determine total number of channel closures during a scan.

How scan count works with the scan list		
Number of steps	Scan count value	Total number of step channel closures
3	1 (default)	3 (each step channel once – 1A05, 1B05, 1C05)
3	2	6 (each step channel twice – 1A05, 1B05, 1C05, then 1A05, 1B05, 1C05)
3	3	9 (each step channel three times – 1A05, 1B05, 1C05 then 1A05, 1B05, 1C05 then 1A05, 1B05, 1C05)
<code>scan.stepcount</code>	<code>scan.scancount</code>	<code>scan.stepcount X scan.scancount</code>

Therefore, scan count represents how many times to loop on the total number of steps in a defined scan list.

If you want to pace the steps of closing and opening channels by detecting a trigger on digio trigger line 3, the following flow chart indicates the sequence of events to achieve this. This example uses the same scan list as above and uses a scan count of 3.

As the flow chart below shows, to transition between start of scan to step 1, step 1 and step 2, step 2 and step 3, a trigger needs to be detected on digio trigger line 3. At the end of step 3, if the scan count has reached three, the scan completes. If the scan count hasn't reached 3 then, loop back around to repeat steps 1 to 3 again.



The following represents a script you can enter using the TSB Embedded page on the web to experiment with the scan points discussed above.

<code>reset()</code>	Reset the instrument
<code>scan.add("1A05, 1B05, 1C05")</code>	Create a three channel scan list
<code>digio.trigger[3].mode = digio.TRIG_FALLING</code>	Detect falling edges on digio trigger line 3
<code>scan.trigger.channel.stimulus = digio.trigger[3].EVENT_ID</code>	Configure digio trigger line 3 to pace the scan sequence
<code>scan.scancount = 3</code>	Set the scan count to 3
<code>print(scan.list())</code>	Show the configured scan list
<code>delay(5)</code>	Delay 5 seconds to allow viewing of scan list
<code>scan.background()</code>	Start the scan and let it run in the background
<code>ScanState, ScanCount, StepCount = scan.state()</code>	Get the current scan state information
<code>print(ScanState, " ", ScanCount, " ", StepCount)</code>	Show the state of scanning
<code>delay(3)</code>	Delay 3 seconds to allow viewing of state
<code>print("state", " scan count", " step count")</code>	Header for output data from for loop
<code>for x = 1, (scan.scancount * scan.stepcount) do</code>	Loop for scan count times step count to step the scan sequence
<code> digio.trigger[3].assert()</code>	Simulate the trigger occurring on digio trigger line 3
<code> ScanState, ScanCount, StepCount = scan.state()</code>	Get the current scan state information
<code> print(ScanState, " ", ScanCount, " ", StepCount)</code>	Show the state of scanning
<code> delay(3)</code>	Delay 3 seconds to allow viewing of state
<code>end</code>	

On the TSB Embedded page, click **Save Script** to save the script. For example, name the script ScanExample. In the User Scripts list, select ScanExample. Click **Run**. The output of the script is:

<code>Init) OPEN...</code>	
<code> 1) STEP: 1A05</code>	
<code> CLOSE: 1A05</code>	
<code> 2) STEP: 1B05</code>	
<code> OPEN: 1A05</code>	
<code> CLOSE: 1B05</code>	
<code> 3) STEP: 1C05</code>	
<code> OPEN: 1B05</code>	
<code> CLOSE: 1C05</code>	
<code>2.000000000e+000</code>	<code>1.000000000e+000</code> <code>0.000000000e+000</code>
<code>state</code>	<code>scan count</code> <code>step count</code>
<code>2.000000000e+000</code>	<code>1.000000000e+000</code> <code>1.000000000e+000</code>
<code>2.000000000e+000</code>	<code>1.000000000e+000</code> <code>2.000000000e+000</code>
<code>2.000000000e+000</code>	<code>1.000000000e+000</code> <code>3.000000000e+000</code>
	Shows scan is running (state = 2) and no steps have completed for scan count of 1
	Shows scan is running (state = 2) and just completed step 1 for scan count of 1
	Shows scan is running (state = 2) and just completed step 2 for scan count of 1
	Shows scan is running (state = 2) and just completed step 3 for scan count of 1

2.000000000e+000	2.000000000e+000	1.000000000e+000	Shows scan is running (state = 2) and just completed step 1 for scan count of 2
2.000000000e+000	2.000000000e+000	2.000000000e+000	Shows scan is running (state = 2) and just completed step 2 for scan count of 2
2.000000000e+000	2.000000000e+000	3.000000000e+000	Shows scan is running (state = 2) and just completed step 3 for scan count of 2
2.000000000e+000	3.000000000e+000	1.000000000e+000	Shows scan is running (state = 2) and just completed step 1 for scan count of 3
2.000000000e+000	3.000000000e+000	2.000000000e+000	Shows scan is running (state = 2) and just completed step 2 for scan count of 3
6.000000000e+000	3.000000000e+000	3.000000000e+000	Shows scan has completed (state = 6) and just completed step 3 for scan count of 3

NOTE

For more examples of using scanning and triggering, see the Models 707B and 708B User's Manual, section "Using a Series 2600 with your Model 707B or 708B."

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 Model 707B or 708B 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 <code>writetbit</code> and <code>writeport</code> 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	<ul style="list-style-type: none"> The programmed state of the line determines if the behavior is similar to <code>RisingA</code> or <code>RisingM</code> High similar to <code>RisingA</code> Low similar to <code>RisingM</code>
Rising A	High	Low	Rising	

Trigger mode	Output		Input	Notes
	Unasserted	Asserted	Detects	
RisingM	Low	High	None	
Synchronous	High latching	Low	Falling	<ul style="list-style-type: none"> Behaves similar to SynchronousA Trigger overrun detection is disabled To mirror the SynchronousA trigger mode, set the pulse duration to 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 [Instrument Control Library \(ICL\) command reference](#) (on page 7-8).

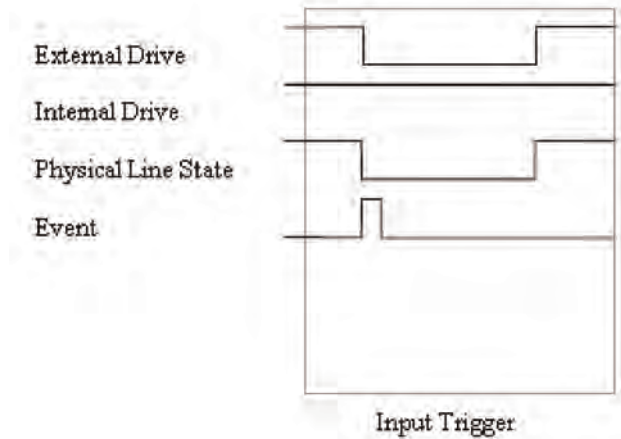
NOTE

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

Falling edge trigger mode

The falling edge trigger mode generates low pulses and detects all falling edges. The following graphic illustrates the characteristics for the falling edge input trigger.

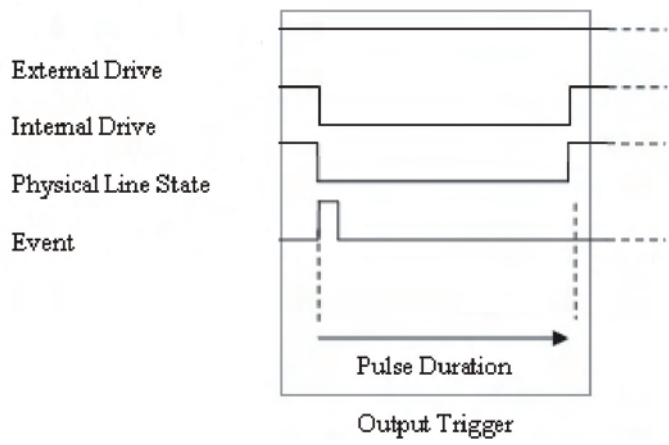
Figure 59: Falling edge input trigger



Input characteristics:

- Detects all falling edges as input triggers

Figure 60: Falling edge output trigger



Output characteristics:

- When the trigger is asserted, it generates a low pulse for the programmed pulse duration.

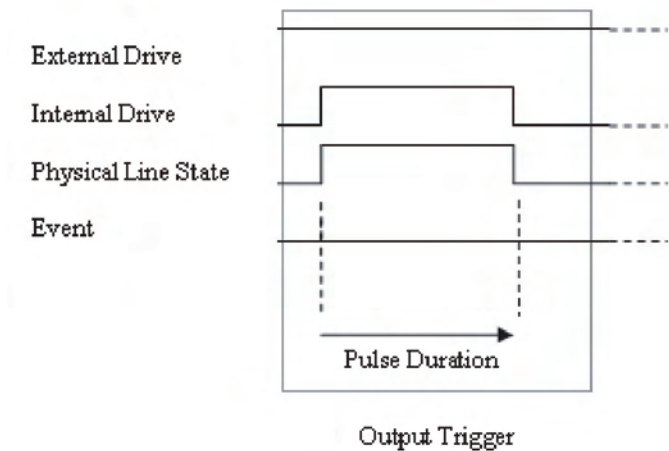
Rising edge master trigger mode

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

NOTE

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

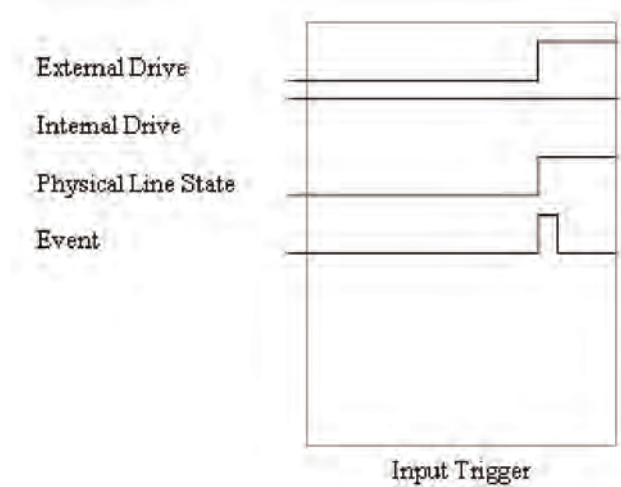
Figure 61: RisingM output trigger



Rising edge acceptor trigger mode

The rising edge acceptor trigger mode (RisingA) generates a low pulse and detects rising edge pulses. The following graphic displays the RisingA input trigger.

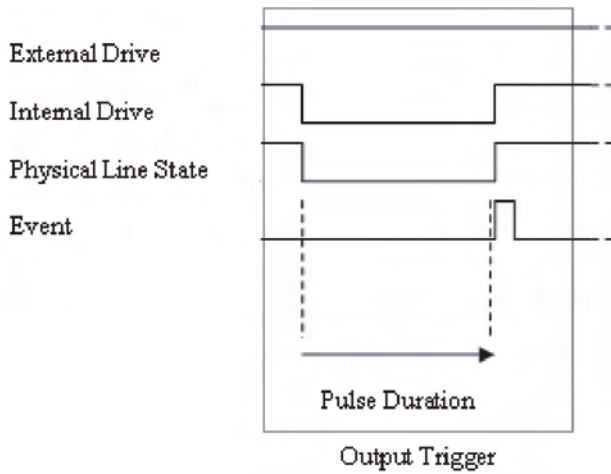
Figure 62: RisingA input trigger



Input characteristics:

- All rising edges generate an input event.

Figure 63: RisingA output trigger



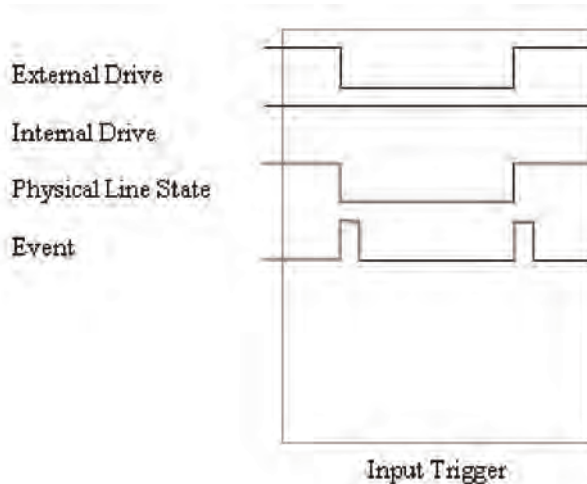
Output characteristics:

- When the trigger is asserted, generates a low pulse that is similar to the falling edge trigger mode.

Either edge trigger mode

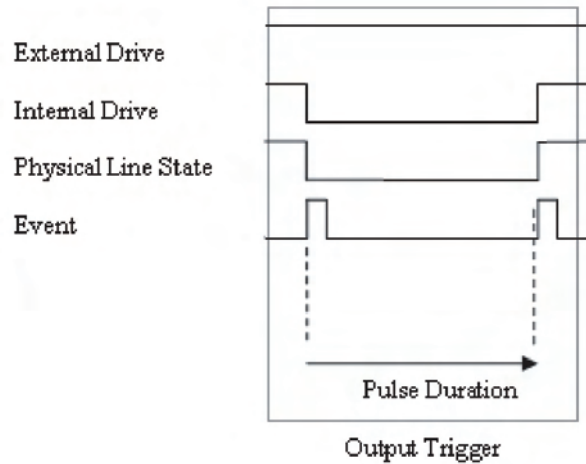
The either edge trigger mode generates a low pulse and detects both rising and falling edges.

Figure 64: Either edge input trigger



Input characteristics:

- All rising or falling edges generate an input trigger event

Figure 65: Either edge output trigger**Output characteristics:**

- When the trigger is asserted, it generates a low pulse that is similar to the falling edge trigger mode.

Understanding synchronous triggering modes

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

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

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

Synchronous master trigger mode

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

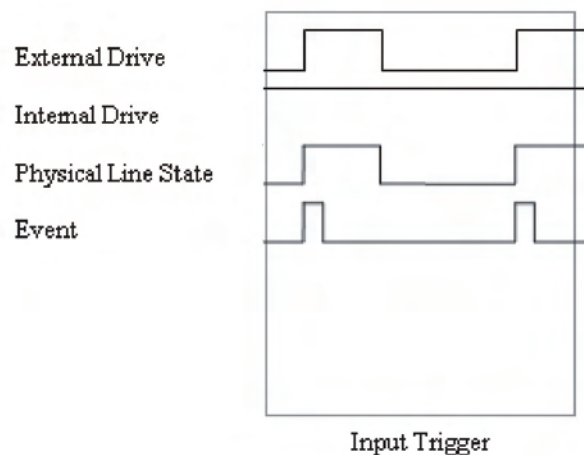
In this mode, the output trigger consists of a low pulse. All non-Keithley instruments attached to the synchronization line in a trigger mode equivalent to SynchronousA must latch the line low during the pulse duration.

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

NOTE

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

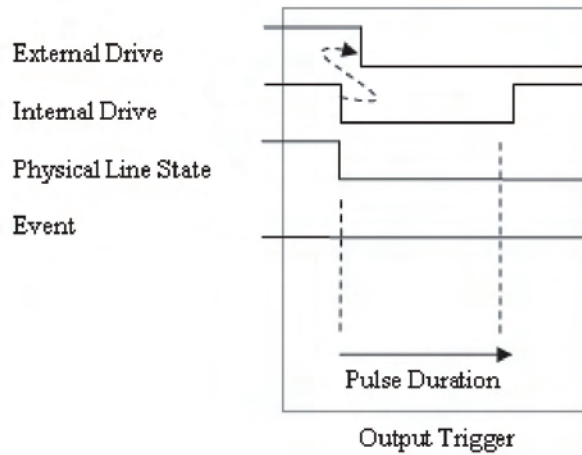
Figure 66: SynchronousM input trigger



Input characteristics:

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

Figure 67: SynchronousM output trigger



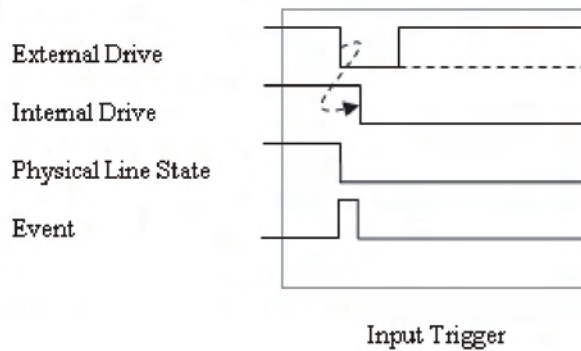
Output characteristics:

- When the trigger is asserted, it generates a low pulse that is similar to the Falling Edge trigger mode

Synchronous acceptor trigger mode

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

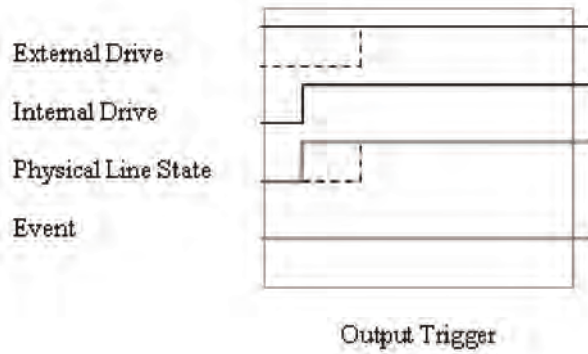
Figure 68: SynchronousA input trigger



Input characteristics:

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

Figure 69: SynchronousA output trigger



Output characteristics:

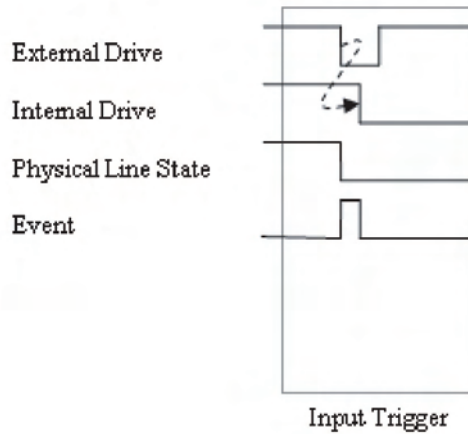
- The physical line state does not change until all drives (internal and external) release the line.

Synchronous trigger mode

The synchronous trigger mode is a combination of SynchronousA and SynchronousM trigger modes.

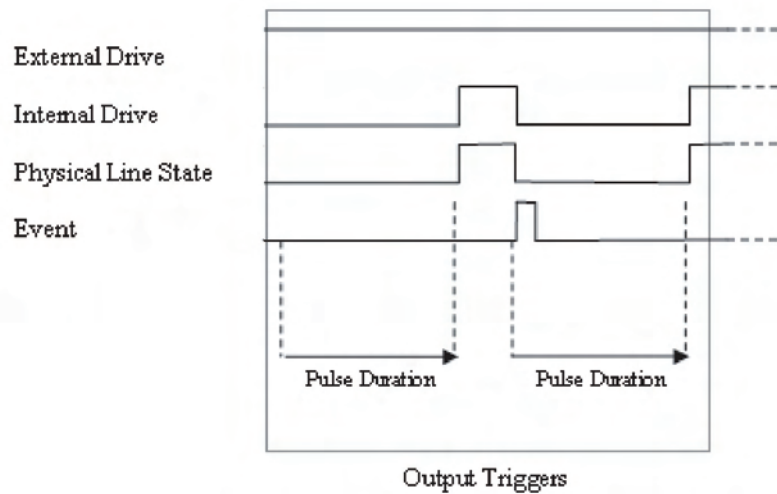
The SynchronousA and SynchronousM trigger modes provide additional flexibility.

Figure 70: Synchronous input trigger



Input characteristics:

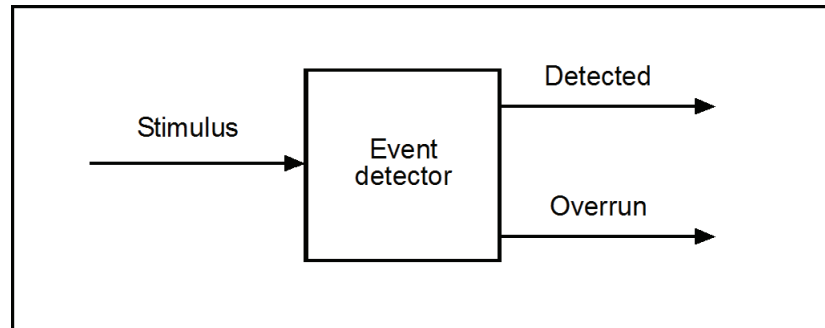
- The falling edge generates an input event and latches the internal drive low.

Figure 71: Synchronous output trigger**Output characteristics:**

- When the trigger is asserted, it generates 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 trigger is asserted.

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 72: 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 ICL commands provide additional information on available blenders:

```
trigger.blender\[N\].clear\(\) (on page 7-201)  
trigger.blender\[N\].orenable (on page 7-202)  
trigger.blender\[N\].overrun (on page 7-203)  
trigger.blender\[N\].stimulus\[M\] (on page 7-204)  
trigger.blender\[N\].wait\(\) (on page 7-205)
```

Theory of operations

In this section:

Models 707B and 708B theory of operations overview 4-1
 Mainframe 4-1
 Front panel 4-6

Models 707B and 708B theory of operations overview

The Models 707B and 708B are composed of several components assembled into an aluminum frame. The Model 707B supports the operation of up to six relay matrix cards; the Model 708B supports the operation of a single relay matrix card. Each of the components is briefly described below.

Mainframe

Figure 73: Model 707B mainframe block diagram

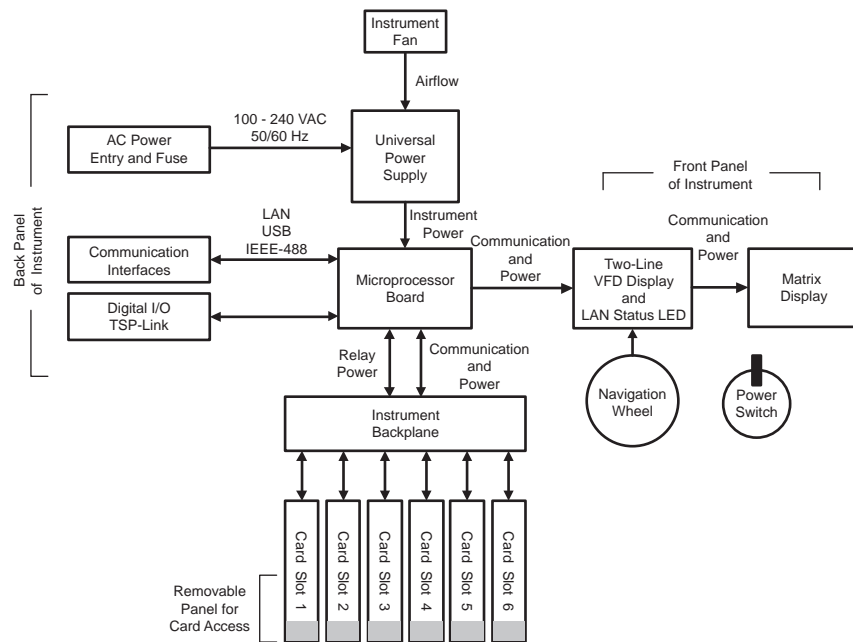
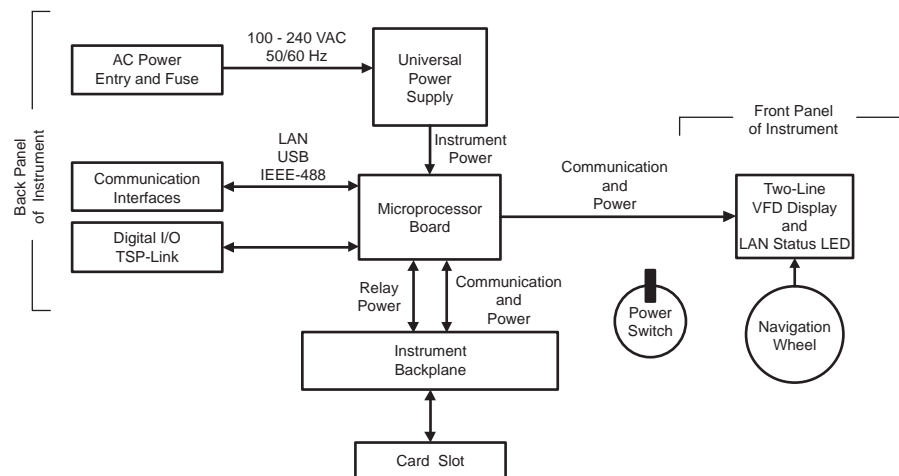


Figure 74: Model 708B mainframe block diagram



Important test system safety information

This product is sold as a stand-alone instrument that may become part of a system that could contain hazardous voltages and energy sources. It is the responsibility of the test system designer, integrator, installer, maintenance personnel, and service personnel to make sure the system is safe during use and is operating properly.

You must also realize that in many test systems a single fault, such as a software error, may output hazardous signal levels even when the system indicates that there is no hazard present.

It is important that you consider the following factors in your system design and use:

- The international safety standard IEC 61010-1 defines voltages as hazardous if they exceed 30 V rms and 42.4 V peak or 60 V dc for equipment rated for dry locations. Keithley Instruments, Inc. products are only rated for dry locations.
- Read and comply with the specifications of all instruments in the system. The overall allowed signal levels may be constrained by the lowest rated instrument in the system. For example, if you are using a 500 volt power supply with a 300 V dc rated switch, the maximum allowed voltage in the system is only 300 V dc.
- Make sure any test fixture connected to the system protects the operator from contact with hazardous voltages, hot surfaces, or sharp objects. This may be accomplished by shields, barriers, insulation, safety interlocks, or the like.
- Cover the device under test (DUT) to protect an operator from flying debris in the event of a system or DUT failure.

- Double insulate all electrical connections that an operator could touch. Double insulation ensures the operator is still protected even if one insulation layer fails. Refer to IEC 61010-1 for specific requirements.
- Make sure all connections are behind a locked cabinet door or other barrier. This protects the system operator from accidentally removing a connection by hand and exposing hazardous voltages.
- Use high reliability fail-safe interlock switches to disconnect power sources when a test fixture cover is opened.
- Where possible, use automatic handlers so operators are not required to access the DUT or other potentially hazardous areas.
- Provide training to all users of the system so they understand all potential hazards and know how to protect themselves from injury.
- In many systems, during power on, the outputs may be in an unknown state until they are properly initialized. Make sure the design can tolerate this situation without causing operator injury or hardware damage.

Of course, read and follow all safety warnings provided with the specific instruments to keep system users safe.

For Model 707B or 708B, also see [Safety Precautions](#) (on page 1-1).

Instrument fan (Model 707B only)

The Model 707B includes an internal fan that operates continuously when power is applied to the mainframe. The fan keeps the internal power supply and other electronics cool during operation.

AC power entry

The Model 707B or 708B is powered from standard AC mains supply through a power entry card that is located on the rear panel of the instrument.

Universal power supply

All the power for the internal electronics and relay cards is provided by a switched-mode power supply which is located inside the mainframe. The power supply offers improved power efficiency compared to earlier versions of the Models 707A and 708A products.

Total power consumption requirements are listed on the rear panel of the instrument.

Microprocessor board

At the heart of the Model 707B or 708B instrument is a microprocessor board that processes all communications and generates power supplies. It outputs information and status messages, and executes operational commands supplied by the operator.

For information on the commands that can be sent to the instrument, see [Remote Commands](#) (on page 5-1).

Communication interfaces

You can operate the Model 707B or 708B over one of several standard remote interfaces. The rear panel includes connectors for these communication interfaces:

- Universal Serial Bus (USB)
- Local area network (LAN)
- General Purpose Interface Bus (GPIB or IEEE-488)

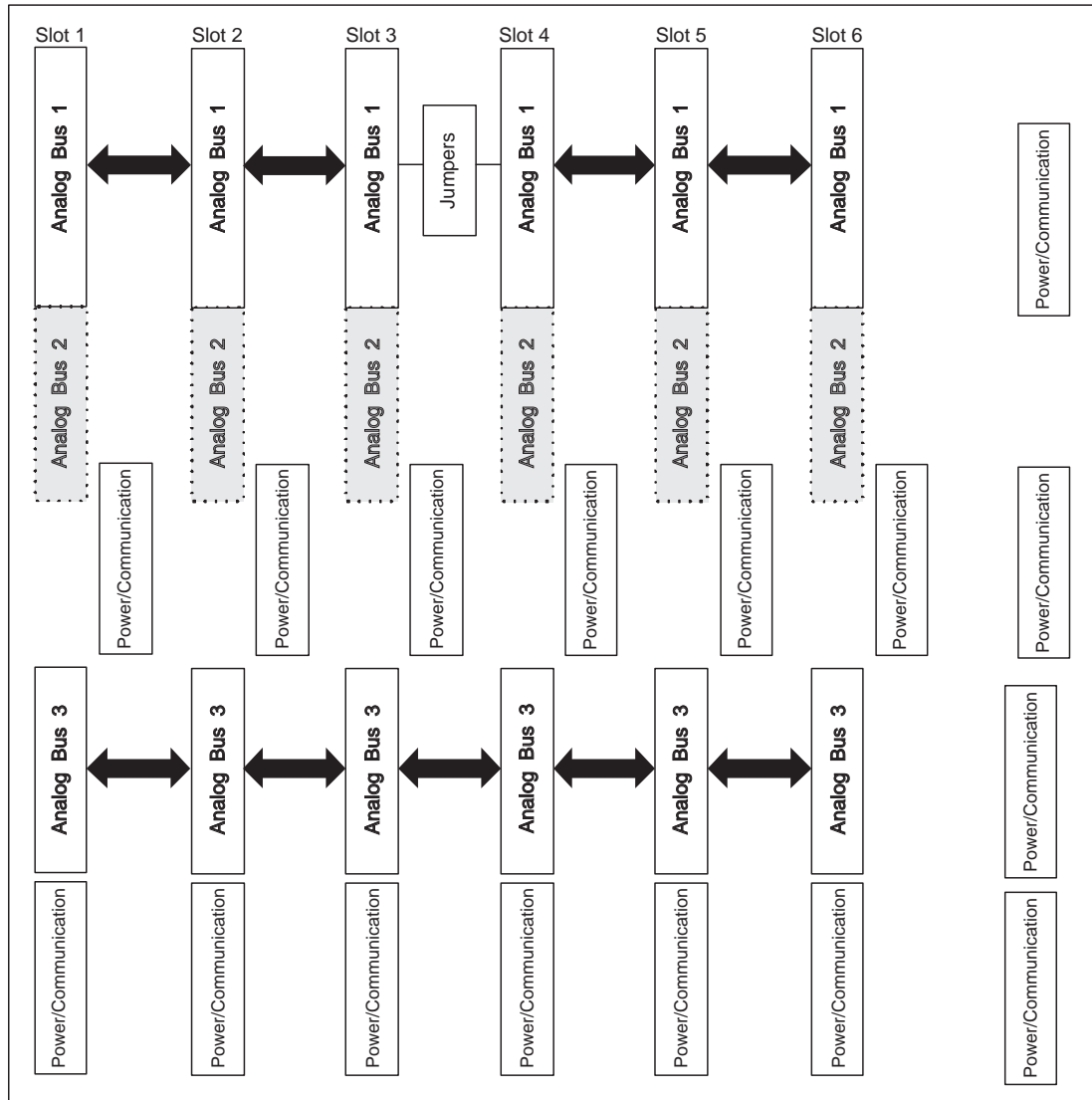
Trigger and control interfaces

You can use rear panel connections to control external digital circuits and instruments that are connected with TSP. See [Digital I/O port](#) (on page 2-7) and [Connect the TSP-Link cable](#) (on page 2-6).

Backplane


The Model 707B or 708B backplane is the interface between the installed relay cards. The relay card interfaces with the backplane through a communication and power connector and possibly an expansion bus for signal routing.


Figure 75: Model 707B backplane



Front panel

The front panel of the Keithley Instruments Model 707B or 708B contains the following items:



- The display
- The crosspoint display (Model 707B only)
- The keys and navigation wheel 
- The LAN status indicator
- The POWER button

You can use the keys, displays, and the navigation wheel  to change the selected channel or channel pattern. You can also use them to access, view, and edit the menu items. The crosspoint display on the Model 707B shows you which channels are opened and closed.

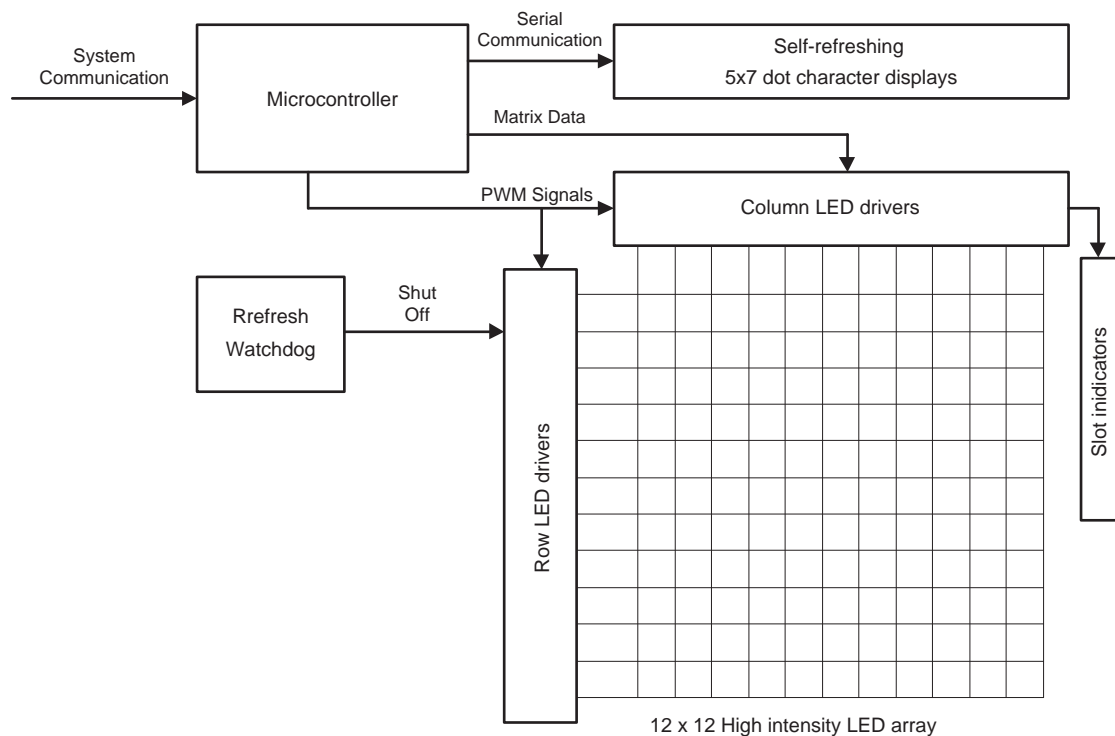
The small display is a two-line vacuum fluorescent display (VFD).

Upper crosspoint display (Model 707B only)

The larger upper display on the Model 707B mainframe provides a visual means of determining the relay status of each card slot in the instrument at any given time. It also shows the label names for each row and column of the given slot.

The crosspoint display may be operated from the navigation wheel  to scroll through the slots displaying the status. You can use the navigation wheel  to select channels for relay open or closures or other actions.

For details on indicators and additional options, see [Crosspoint display \(Model 707B only\)](#) (on page 2-15).

Figure 76: Model 707B crosspoint display block diagram

Description of the crosspoint display components

The front panel of the Model 707B is composed of two main display components: a vacuum fluorescent display (VFD) and a 12 x 12 LED matrix display, called the crosspoint display, that shows the relay status for each of the slots.

The block diagram shows only the matrix display. Its constituent components are described below.

Microcontroller

The microcontroller receives commands from the main controller and updates the row and column labels. It also generates the pulse-width modulated (PWM) signals for operating the 12 x 12 LED matrix.

Row and column labels

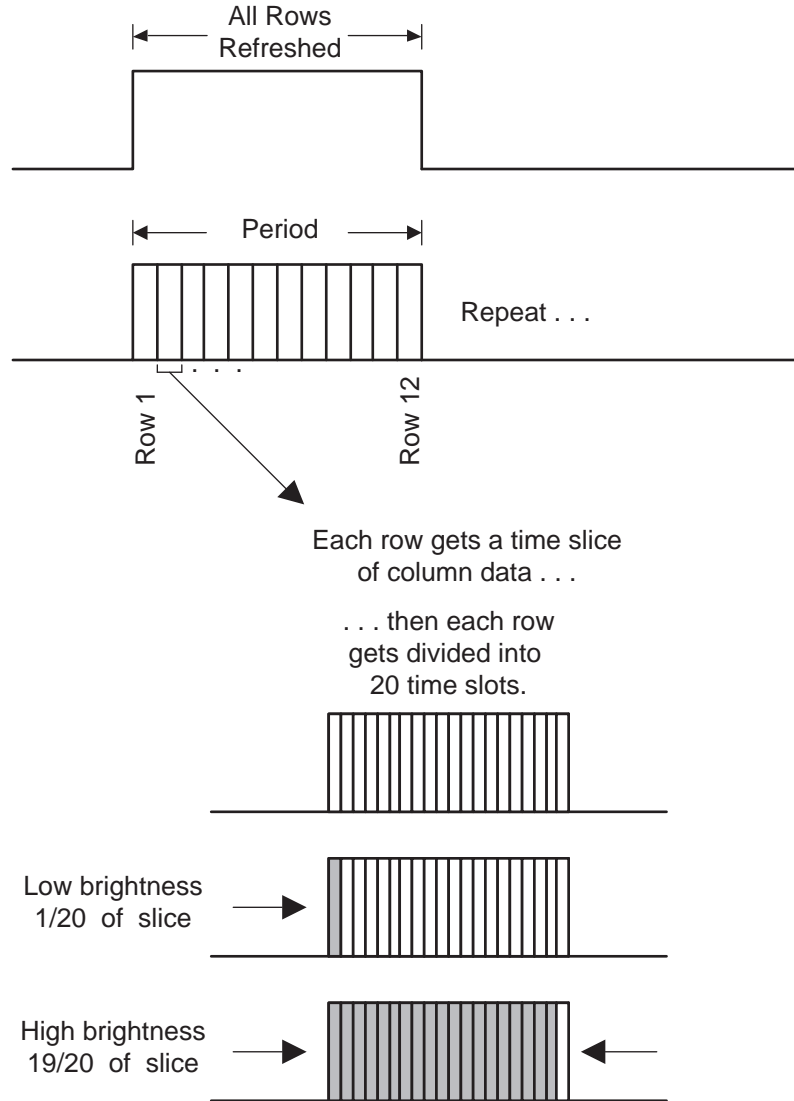
The row and column labels are updated by writing to the devices serially. They handle their own brightness and refresh controls, alleviating the main microcontroller from performing these tasks. As a result, these are only updated when the data changes and a command from the instrument is issued to update the label.

12 x 12 LED matrix

The 12 x 12 matrix array of LEDs is driven by PWM signals that cause the intensity of the LED to be bright or dim based on the appropriate setting from the instrument.

Each row and column is updated sequentially. The entire display has a refresh rate of approximately 60 Hz. The time domain details of display refreshing are shown below.

Figure 77: Model 707B display refresh



Slot indicators

The slot LEDs are treated the same as the matrix LEDs, as additional 13th and 14th columns.

Display watchdog

Once the display tasks are started, if the microcontroller stops refreshing, the watchdog circuit shuts off the column drivers to preserve the life of the LEDs.

The display watchdog is part of the standard display test (see [Testing the display, keys, and channel matrix](#) (on page 8-9) for information).

If the watchdog shuts down the display, the first several row labels show "SLOT GRID LEDS OFF."

Description of basic display operation

This display board is a subordinate device to the microprocessor board.

The microprocessor board communicates commands to change the state of the matrix or update the row and column labels.

The display board processes these commands independently. It also refreshes the display and performs other tasks. When a display command is issued, the microcontroller processes it to update the display. The task of refreshing rows and PWM generation is done in an interrupt-driven routine to ensure predictable timing.

Remote commands

In this section:

Introduction to remote operation	5-1
About ICL commands.....	5-4
Overview of instrument drivers.....	5-18
Migrating from Models 707A and 708A	5-21

Introduction to remote operation

Keithley Instruments Test Script Processor-enabled instruments 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.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without a controller. You can load a series of instrument control commands into the instrument and 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 particular 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 instrument control commands. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

Instrument control 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.

Functions

Function-based commands control actions or activities. For example, performing a channel closure is a function. A function-based command is not always directly related to instrument operation. For example, the `bit.bitand` function will perform a logical AND operation on two numbers.

Each function consists of a function name followed by a set of parenthesis (). If the function does not have a parameter, the parenthesis set is left empty.

Example 1

<code>digio.writeport(15)</code>	Sets digital I/O lines 1, 2, 3, and 4 high
<code>digio.writebit(3, 0)</code>	Sets line 3 to low (0)
<code>reset()</code>	Returns the instrument to its default settings
<code>digio.readport()</code>	Reads the digital I/O port

Example 2

The results of a function-based command can be used directly or can be assigned to variables for later access. The following code saves the value an instrument operator enters from the front panel and prints it.

<code>value = display.inputvalue("+0.00")</code> <code>print(value)</code>	If the operator enters 2.36 from the front panel, the resulting output is: 2.3600000e+00
-------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

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 number of errors in the error queue (`errorqueue.count`).

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

<code>format.data = 3</code>	This attribute sets the format of data printed by other commands. Setting this attribute to 3 sets the print format to double precision floating point format.
------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------

Example 2: Set an attribute using a constant

<code>format.data = format.REAL64</code>	Using the constant <code>REAL64</code> instead of 3 also 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 1: Read an attribute using a function

<pre>print(format.data)</pre>	<p>Reads the data format by passing the attribute to the print function. If the data format is set to 3, the output is:</p> <pre>3.0000000e+00</pre> <p>This shows that the data format is set to double precision floating point.</p>
-------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example 2: Read an attribute using a variable

<pre>fd = format.data</pre>	<p>This reads the data format by assigning the attribute to a variable named fd.</p>
-----------------------------	--------------------------------------------------------------------------------------

Queries

TSP-enabled instruments do not have inherent query commands. Like any other scripting environment, the `print()` command and other related commands generate output in the form of response messages. Each `print()` command creates one response message.

Example

Code	Notes and output
<pre>x = 10 print(x)</pre>	<p>Example of an output response message:</p> <pre>1.0000000e+01</pre> <p>Note that your output might be different if you set your ASCII precision setting to a different value.</p>

Data retrieval commands

You can send data retrieval commands that return a comma-delimited string. For example, `print(memory.available())` returns the amount of memory that is available in the instrument.

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.

Information on scripting and programming

If you need information on using scripts with Model 707B or 708B, see [Fundamentals of scripting for TSP](#) (on page 6-1).

If you need information on using the Lua programming language with Model 707B or 708B, see [Fundamentals of programming for TSP](#) (on page 6-15).

About ICL commands

This section contains an overview of the ICL commands organized into to groups, with a brief description of each group and links to the detailed command descriptions in the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) in this manual.

beeper functions and attributes

The beeper commands allow you to enable or disable the instrument beeper and set the frequency. When enabled, you can set the duration of the beep.

```
beeper.beep()  
beeper.enable (on page 7-9)
```

bit functions

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 logic operations on two numbers. The Test Script Processor (TSP) performs the indicated logic operation on the binary equivalents of the two integers. Bitwise logic operations are performed. This means that the logical AND, OR, or XOR operation is performed on Bit 1 of the first number and Bit 1 of the second number. The logical AND, OR, or XOR operation is performed on Bit 2 of the first number and Bit 2 of the second number. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation 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 of 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) stores all numbers internally as IEEE Std 754 double-precision floating point values. The logical operations work on 32-bit integers. Any fractional bits will be truncated. For numbers larger than 4294967295, only the lower 32 bits will be used.

[bit.bitand\(\)](#) (on page 7-9)
[bit.bitor\(\)](#) (on page 7-10)
[bit.bitxor\(\)](#) (on page 7-11)
[bit.clear\(\)](#) (on page 7-11)
[bit.get\(\)](#) (on page 7-12)
[bit.getfield\(\)](#) (on page 7-13)
[bit.set\(\)](#) (on page 7-14)
[bit.setfield\(\)](#) (on page 7-15)
[bit.test\(\)](#) (on page 7-16)
[bit.toggle\(\)](#) (on page 7-17)

channel functions and attributes**About channel commands****Channel identifiers**

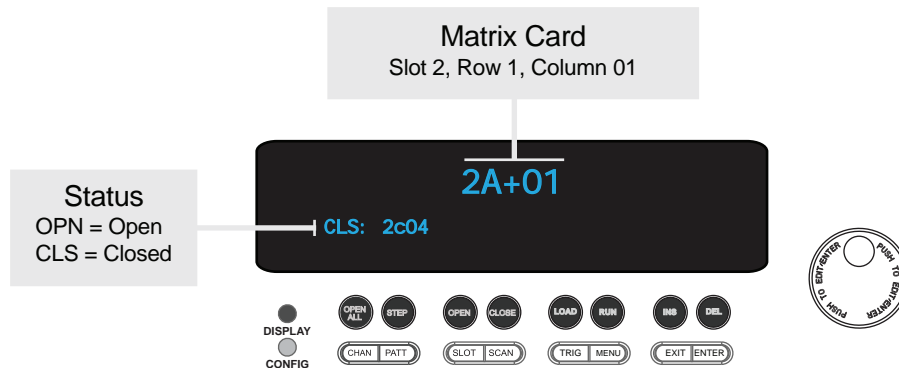
The channels on the matrix cards are referred to by their slot, bank, row, and column numbers:

- **Slot number:** The number of the slot in which the card is installed.
- **Bank:** 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 counting sequence is: 98, 99, A0, A1, ..., A8, A9, B0, B1, ...

Matrix channel examples

Reference	Slot	Bank	Row	Column
1A05	1	N/A	1	05
1C05	1	N/A	3	05
3C12	3	N/A	3	12

Figure 78: Matrix card display showing channel identifier



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 ICL command determines the number and order of these returned items.

When the *channelList* parameter for these functions is "slotX", 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 Model 707B. The Model 708B has only one slot, thus "allslots" is the same as "slot1" for this model. 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
```

The channel functions and attributes are:

[channel.clearforbidden\(\)](#) (on page 7-18)
[channel.close\(\)](#) (on page 7-19)
[channel.connectrule](#) (on page 7-20)
[channel.connectsequential](#) (on page 7-21)
[channel.createspecifier\(\)](#) (on page 7-23)
[channel.exclusiveclose\(\)](#) (on page 7-24)
[channel.exclusiveslotclose\(\)](#) (on page 7-25)
[channel.getclose\(\)](#) (on page 7-27)
[channel.getcount\(\)](#) (on page 7-28)
[channel.getdelay\(\)](#) (on page 7-29)
[channel.getforbidden\(\)](#) (on page 7-30)
[channel.getlabel\(\)](#) (on page 7-31)
[channel.getlabelcolumn\(\)](#) (on page 7-32)
[channel.getlabelrow\(\)](#) (on page 7-33)
[channel.getstate\(\)](#) (on page 7-34)
[channel.gettype\(\)](#) (on page 7-35)
[channel.open\(\)](#) (on page 7-35)
[channel.pattern.catalog\(\)](#) (on page 7-37)
[channel.pattern.delete\(\)](#) (on page 7-38)
[channel.pattern.getimage\(\)](#) (on page 7-38)
[channel.pattern.setimage\(\)](#) (on page 7-39)
[channel.pattern.snapshot\(\)](#) (on page 7-41)
[channel.reset\(\)](#) (on page 7-43)
[channel.setdelay\(\)](#) (on page 7-44)
[channel.setforbidden\(\)](#) (on page 7-45)
[channel.setlabel\(\)](#) (on page 7-46)
[channel.setlabelcolumn\(\)](#) (on page 7-47)
[channel.setlabelrow\(\)](#) (on page 7-49)

createconfigscript function

This function captures the present settings of the instrument.

[createconfigscript\(\)](#) (on page 7-50)

dataqueue functions and attributes

Use the dataqueue 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 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.

[dataqueue.add\(\)](#) (on page 7-51)
[dataqueue.CAPACITY](#) (on page 7-52)
[dataqueue.clear\(\)](#) (on page 7-52)
[dataqueue.count](#) (on page 7-53)
[dataqueue.next\(\)](#) (on page 7-54)

delay function

This function is used to hold up 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.

[delay\(\)](#) (on page 7-55)

digio functions and attributes

The digital I/O port of the Model 707B or 708B 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 in [Digital I/O port](#) (on page 2-7) for additional information.

There are commands to read and write to each individual bit, and commands to read and write to the entire port. Use the following code fragment to write to one bit of the digital I/O port. The I/O bit is then read and the state is returned to the computer, where it can be processed or displayed.

Example: Write to one bit of the digital I/O port	
Command	Description
<code>digio.writebit(4, 0)</code>	Writes a 0 to Bit 4
<code>data = digio.readbit(4)</code>	Reads the value at Bit 4
<code>print(data)</code>	Generates a response message

[digio.readbit\(\)](#) (on page 7-55)
[digio.readport\(\)](#) (on page 7-56)
[digio.trigger\[N\].assert\(\)](#) (on page 7-57)
[digio.trigger\[N\].clear\(\)](#) (on page 7-57)
[digio.trigger\[N\].EVENT_ID](#) (on page 7-58)
[digio.trigger\[N\].mode](#) (on page 7-58)
[digio.trigger\[N\].overrun](#) (on page 7-60)
[digio.trigger\[N\].pulsewidth](#) (on page 7-60)
[digio.trigger\[N\].release\(\)](#) (on page 7-61)
[digio.trigger\[N\].reset\(\)](#) (on page 7-62)
[digio.trigger\[N\].stimulus](#) (on page 7-62)
[digio.trigger\[N\].wait\(\)](#) (on page 7-64)
[digio.writebit\(\)](#) (on page 7-64)
[digio.writeport\(\)](#) (on page 7-65)
[digio.writeprotect](#) (on page 7-66)

display functions and attributes

[display.clear\(\)](#) (on page 7-67)
[display.getannunciators\(\)](#) (on page 7-67)
[display.getcursor\(\)](#) (on page 7-68)
[display.getlastkey\(\)](#) (on page 7-69)
[display.gettext\(\)](#) (on page 7-70)
[display.inputvalue\(\)](#) (on page 7-73)
[display.loadmenu.add\(\)](#) (on page 7-74)
[display.loadmenu.catalog\(\)](#) (on page 7-76)
[display.loadmenu.delete\(\)](#) (on page 7-76)
[display.locallockout](#) (on page 7-77)
[display.menu\(\)](#) (on page 7-77)
[display.prompt\(\)](#) (on page 7-78)
[display.screen](#) (on page 7-80)
[display.sendkey\(\)](#) (on page 7-80)
[display.setcursor\(\)](#) (on page 7-81)
[display.settext\(\)](#) (on page 7-82)
[display.trigger.EVENT_ID](#) (on page 7-84)
[display.waitkey\(\)](#) (on page 7-84)

errorqueue functions and attributes

When errors occur, the error messages are placed in the error queue. Use error queue commands to request error message information.

[errorqueue.clear\(\)](#) (on page 7-85)
[errorqueue.count](#) (on page 7-86)
[errorqueue.next\(\)](#) (on page 7-86)

eventlog functions and attributes

The functions and attributes in this group control the event log.

[eventlog.all\(\)](#) (on page 7-87)
[eventlog.clear\(\)](#) (on page 7-88)
[eventlog.count](#) (on page 7-89)
[eventlog.enable](#) (on page 7-90)
[eventlog.next\(\)](#) (on page 7-90)
[eventlog.overwritemethod](#) (on page 7-91)

exit function

Use this function from in your script to terminate it.

[exit\(\)](#) (on page 7-92)

format attributes

These attributes determine how information is formatted when it is printed using the `print()`, `printbuffer()`, and `printnumber()` functions.

[format.asciiprecision](#) (on page 7-93)
[format.byteorder](#) (on page 7-93)
[format.data](#) (on page 7-94)

gpib attribute

This attribute describes the GPIB address.

[gpib.address](#) (on page 7-97)

lan functions and attributes

The LAN commands has 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()` in order for the configuration settings to take effect.

The `lan.status.*` commands help you determine the status of the LAN.

The `lan.trigger[N].*` commands allows 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.

[lan.applysettings\(\)](#) (on page 7-98)
[lan.config.dns.address\[N\]](#) (on page 7-98)
[lan.config.dns.domain](#) (on page 7-99)
[lan.config.dns.dynamic](#) (on page 7-100)
[lan.config.dns.hostname](#) (on page 7-101)
[lan.config.dns.verify](#) (on page 7-101)
[lan.config.gateway](#) (on page 7-102)
[lan.config.ipaddress](#) (on page 7-104)
[lan.config.method](#) (on page 7-104)
[lan.config.subnetmask](#) (on page 7-105)
[lan.lxidomain](#) (on page 7-106)
[lan.nagle](#) (on page 7-106)
[lan.reset\(\)](#) (on page 7-107)
[lan.restoredefaults\(\)](#) (on page 7-107)
[lan.status.dns.address\[N\]](#) (on page 7-108)
[lan.status.dns.name](#) (on page 7-109)
[lan.status.duplex](#) (on page 7-109)
[lan.status.gateway](#) (on page 7-110)
[lan.status.ipaddress](#) (on page 7-110)
[lan.status.macaddress](#) (on page 7-111)
[lan.status.port.dst](#) (on page 7-111)
[lan.status.port.rawsocket](#) (on page 7-112)
[lan.status.port.telnet](#) (on page 7-112)
[lan.status.port.vxill](#) (on page 7-113)
[lan.status.speed](#) (on page 7-113)
[lan.status.subnetmask](#) (on page 7-114)
[lan.trigger\[N\].assert\(\)](#) (on page 7-114)
[lan.trigger\[N\].clear\(\)](#) (on page 7-115)
[lan.trigger\[N\].connect\(\)](#) (on page 7-116)
[lan.trigger\[N\].connected](#) (on page 7-117)
[lan.trigger\[N\].disconnect\(\)](#) (on page 7-118)
[lan.trigger\[N\].EVENT_ID](#) (on page 7-118)
[lan.trigger\[N\].ipaddress](#) (on page 7-119)
[lan.trigger\[N\].mode](#) (on page 7-119)
[lan.trigger\[N\].overrun](#) (on page 7-121)
[lan.trigger\[N\].protocol](#) (on page 7-122)

[lan.trigger\[N\].pseudostate](#) (on page 7-122)
[lan.trigger\[N\].stimulus](#) (on page 7-123)
[lan.trigger\[N\].wait\(\)](#) (on page 7-125)

localnode functions and attributes

Commands that allow you to set and read from the local node.

[localnode.define.*](#) (on page 7-125)
[localnode.description](#) (on page 7-126)
[localnode.execute\(\)](#) (on page 7-127)
[localnode.getglobal\(\)](#) (on page 7-128)
[localnode.prompts](#) (on page 7-129)
[localnode.prompts4882](#) (on page 7-130)
[localnode.reset\(\)](#) (on page 7-131)
[localnode.revision](#) (on page 7-131)
[localnode.serialno](#) (on page 7-132)
[localnode.setglobal\(\)](#) (on page 7-133)
[localnode.showerrors](#) (on page 7-133)

make accessor functions

Use these functions to create functions to get and set attribute values.

[makegetter\(\)](#) (on page 7-134)
[makesetter\(\)](#) (on page 7-135)

memory functions

Check the amount of memory that is available or used in the instrument

[memory.available\(\)](#) (on page 7-135)
[memory.used\(\)](#) (on page 7-137)

opc function

This function sets the Operation Complete status bit when all overlapped commands are completed.

[opc\(\)](#) (on page 7-137)

print functions

Return data from the instrument.

[print\(\)](#) (on page 7-138)
[printbuffer\(\)](#) (on page 7-138)
[printnumber\(\)](#) (on page 7-140)

reset function

Resets commands to their default settings.

[reset\(\)](#) (on page 7-140)

scan functions and attributes

The scan functions and attributes allow you to set up scanning over the remove interface.

[scan.abort\(\)](#) (on page 7-141)
[scan.add\(\)](#) (on page 7-142)
[scan.addimagestep\(\)](#) (on page 7-143)
[scan.background\(\)](#) (on page 7-144)
[scan.bypass](#) (on page 7-145)
[scan.create\(\)](#) (on page 7-146)
[scan.execute\(\)](#) (on page 7-147)
[scan.list\(\)](#) (on page 7-148)
[scan.mode](#) (on page 7-150)
[scan.reset\(\)](#) (on page 7-150)
[scan.scancount](#) (on page 7-151)
[scan.state\(\)](#) (on page 7-152)
[scan.stepcount](#) (on page 7-153)
[scan.trigger.arm.clear\(\)](#) (on page 7-153)
[scan.trigger.arm.set\(\)](#) (on page 7-154)
[scan.trigger.arm.stimulus](#) (on page 7-154)
[scan.trigger.channel.clear\(\)](#) (on page 7-156)
[scan.trigger.channel.set\(\)](#) (on page 7-156)
[scan.trigger.channel.stimulus](#) (on page 7-157)
[scan.trigger.clear\(\)](#) (on page 7-158)

script functions and attributes

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, and run scripts.

[script.anonymous](#) (on page 7-159)
[script.delete\(\)](#) (on page 7-159)
[script.new\(\)](#) (on page 7-160)
[script.restore\(\)](#) (on page 7-162)
[script.user.catalog\(\)](#) (on page 7-163)
[scriptVar.autorun](#) (on page 7-163)
[scriptVar.list\(\)](#) (on page 7-164)
[scriptVar.name](#) (on page 7-164)
[scriptVar.run\(\)](#) (on page 7-166)
[scriptVar.save\(\)](#) (on page 7-166)
[scriptVar.source](#) (on page 7-167)

slot[X] attributes

The slot attributes configure and read the settings of the cards in the slots. You can also set up pseudocards.

```
slot\[X\].idn (on page 7-169)  
slot\[X\].poles.four (on page 7-170)  
slot\[X\].poles.one (on page 7-171)  
slot\[X\].poles.two (on page 7-171)  
slot\[X\].pseudocard (on page 7-172)
```

status functions and attributes

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 7-173)  
status.node\_enable (on page 7-175)  
status.node\_event (on page 7-177)  
status.operation.\* (on page 7-178)  
status.operation.user.\* (on page 7-180)  
status.questionable.\* (on page 7-182)  
status.request\_enable (on page 7-184)  
status.request\_event (on page 7-186)  
status.reset\(\) (on page 7-187)  
status.standard.\* (on page 7-188)  
status.system.\* (on page 7-190)  
status.system2.\* (on page 7-192)  
status.system3.\* (on page 7-194)  
status.system4.\* (on page 7-196)  
status.system5.\* (on page 7-198)
```

timer functions

Use the functions in this group to control the timer. The timer can be used to measure the time it takes to perform various operations. Use the `timer.reset()` function at the beginning of an operation to reset the timer to zero, and then use the `timer.measure.t()` at the end of the operation to measure the elapsed time.

```
timer.measure.t\(\) (on page 7-200)  
timer.reset\(\) (on page 7-201)
```

trigger functions and attributes

Use the trigger functions and attributes to control specific trigger objects.

```
trigger.blender\[N\].clear\(\) (on page 7-201)  
trigger.blender\[N\].EVENT\_ID (on page 7-202)  
trigger.blender\[N\].orenable (on page 7-202)  
trigger.blender\[N\].overrun (on page 7-203)  
trigger.blender\[N\].reset\(\) (on page 7-203)  
trigger.blender\[N\].stimulus\[M\] (on page 7-204)  
trigger.blender\[N\].wait\(\) (on page 7-205)  
trigger.clear\(\) (on page 7-206)  
trigger.EVENT\_ID (on page 7-206)  
trigger.timer\[N\].clear\(\) (on page 7-207)  
trigger.timer\[N\].count (on page 7-207)  
trigger.timer\[N\].delay (on page 7-208)  
trigger.timer\[N\].delaylist (on page 7-209)  
trigger.timer\[N\].EVENT\_ID (on page 7-209)  
trigger.timer\[N\].overrun (on page 7-210)  
trigger.timer\[N\].passthrough (on page 7-211)  
trigger.timer\[N\].reset\(\) (on page 7-211)  
trigger.timer\[N\].stimulus (on page 7-212)  
trigger.timer\[N\].wait\(\) (on page 7-213)  
trigger.wait\(\) (on page 7-214)
```

tsplink functions and attributes

These functions and attributes allow you to set up and work with a system that is connected with TSP-Link.

```
tsplink.group (on page 7-214)  
tsplink.master (on page 7-215)  
tsplink.node (on page 7-215)  
tsplink.readbit\(\) (on page 7-216)  
tsplink.readport\(\) (on page 7-216)  
tsplink.reset\(\) (on page 7-217)  
tsplink.state (on page 7-218)  
tsplink.trigger\[N\].assert\(\) (on page 7-218)  
tsplink.trigger\[N\].clear\(\) (on page 7-219)  
tsplink.trigger\[N\].EVENT\_ID (on page 7-219)  
tsplink.trigger\[N\].mode (on page 7-220)  
tsplink.trigger\[N\].overrun (on page 7-221)  
tsplink.trigger\[N\].pulsewidth (on page 7-222)  
tsplink.trigger\[N\].release\(\) (on page 7-223)  
tsplink.trigger\[N\].stimulus (on page 7-224)  
tsplink.trigger\[N\].wait\(\) (on page 7-225)  
tsplink.writebit\(\) (on page 7-226)  
tsplink.writeport\(\) (on page 7-226)  
tsplink.writeprotect (on page 7-227)
```

tspnet functions and attributes

The TSP-Net module provides a simple socket-like programming interface to TSP[®] enabled instruments.

```
tspnet.clear\(\) (on page 7-228)  
tspnet.connect\(\) (on page 7-229)  
tspnet.disconnect\(\) (on page 7-230)  
tspnet.execute\(\) (on page 7-230)  
tspnet.idn\(\) (on page 7-232)  
tspnet.read\(\) (on page 7-232)  
tspnet.readavailable\(\) (on page 7-233)  
tspnet.reset\(\) (on page 7-234)  
tspnet.termination\(\) (on page 7-235)  
tspnet.timeout (on page 7-236)  
tspnet.tsp.abort\(\) (on page 7-236)  
tspnet.tsp.abortonconnect (on page 7-237)  
tspnet.tsp.rhtablecopy\(\) (on page 7-238)  
tspnet.tsp.runscript\(\) (on page 7-239)  
tspnet.write\(\) (on page 7-240)
```

userstring functions

Use the functions in this group to store and retrieve user-defined strings in nonvolatile memory. These strings are stored as key-value pairs. You can use the `userstring` function to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

```
userstring.add\(\) (on page 7-240)  
userstring.catalog\(\) (on page 7-241)  
userstring.delete\(\) (on page 7-242)  
userstring.get\(\) (on page 7-242)
```

waitcomplete function

Allows you to send a command to wait for all overlapped operations in a group to complete.

```
waitcomplete\(\) (on page 7-243)
```

Overview of instrument drivers

To use an instrument connected to a computer, you need to send instrument commands to it to make it do what you want. This can be tedious, as the programmer has to learn the low level syntax and also deal with how their programming language or development environment interfaces to the remote communication interface driver or VISA I/O library. Keithley and most other test and measurement companies supply instrument drivers for their instruments.

One thing a driver can do is deal with cross-coupling issues. If you change a setting on an instrument, this can cause other settings to change without notifying you. For example, if you change the range on an instrument, the resolution can change at the same time. Instrument drivers take care of these issues for you as long as it is feasible. Therefore, a driver would have a call where you pass in the measurement function, range, and resolution. The driver would then take care of sending the correct sequence of commands or generate an error if it was impossible to set the requested values.

For information on finding instrument drivers on the Keithley website, see [Obtaining instrument drivers](#) (on page 5-20).

Instrument driver types

There are several different styles of instrument drivers. Keithley Instruments provides three different instrument drivers for the Models 707B and 708B: 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-n-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 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. There are many VXIPnP drivers available for Agilent, Fluke and Tektronix instruments.

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 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) (<http://www.ivifoundation.org>) defined a set of APIs (Class interfaces) 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 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) (<http://pacificmindworks.com/docs/Making%20the%20Case%20for%20IVI.pdf>).

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 OSX, and Microsoft Windows).

NI maintains a native [LabVIEW driver style guide](http://zone.ni.com/devzone/cda/tut/p/id/3271) (<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 website](http://www.keithley.com) (<http://www.keithley.com>).
2. Select the Support tab.
3. Enter the model number of your instrument.
4. 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 Windows Start menu, under **Keithley Instruments > Model Number** (where Model Number is the instrument model number).

Migrating from Models 707A and 708A

Migrating Model 707A or 708A programs to Model 707B or 708B

This section is intended to assist you if you are migrating existing programs from the Model 707A or 708A to use the TSP programming syntax on the Model 707B or 708B.

The Model 707A and 708A instruments use device-dependent command (DDC) programming. The Model 707B or 708B use the TSP scripting model. This section provides you with:

- Command syntax differences
- A brief explanation of some Model 707A and 708A model-specific terminology and its Model 707B or 708B equivalent

NOTE

Users who want to run programs using Model 707A or 708A DDC commands can run in Model 707A or 708A compatibility mode. See [Using Models 707A and 708A compatibility mode](#) (on page B-1).

Platform differences

When writing a script for the Model 707B or 708B instrument, consider the following platform differences.

	Model 707A or 708A	Model 707B or 708B
Execution host	A computer sends commands over GPIB from a user-generated program such as Visual Basic, C# or C/C++	A computer sends either commands or user-generated scripts over GPIB, USB, or Ethernet. Scripts can also be run from the front panel of the Model 707B or 708B.
Command structure	Single ASCII capital letter commands, followed by argument if necessary.	Descriptive, word-based command and argument structure using modern dot notation format.
Store program flow control	Not possible; instrument can only step sequentially through a stored relay setup in response to an external trigger.	TSP scripting environment allows access to all Lua program control structure operations, such as for-next, if-then-else, while-do, and repeat-until.

Execution host

One of the most significant differences between the Model 707A or 708A and Model 707B or 708B is the ability to store user scripts on the instrument. Once loaded onto the instrument, these scripts can be run without connection to a computer. This is different from the Model 707A or 708A, which could only iterate through an existing stored relay setup if there was no connection to a computer. The ability of the Model 707B or 708B to run scripts that are stored locally results in fast execution time and no communications bottlenecks between the instrument and the computer.

Command structure

The DDC structure of the Model 707A or 708A instrument is based on a single ASCII capital letter with possible alphanumeric arguments and terminated with the "X" execute command.

The Model 707B or 708B instrument uses a modern dot notation format that logically organizes commands by family, subset 1, subset 2 – optional, and argument. For example:

```
channel.open("allslots")
```

Stored program flow control

DDC programming depended on an additional programming environment to send commands to the 707A or 708A instrument. This provided options for program flow control, but at the expense of added complexity.

The TSP scripting environment uses the Lua programming language, providing a robust and cohesive flow control framework.

DDC to ICL command equivalencies

In many cases, DDC commands have a single corresponding ICL equivalent command that performs the same or a similar function.

In some cases, more than one ICL command is needed to provide the same functionality of its equivalent DDC counterpart because of the expanded capabilities and additional flexibility of the TSP scripting environment.

The following sections describe the equivalent commands or sets of commands.

Commands with one-to-one equivalents

The commands in the following table are equivalent.

Commands with one-to-one equivalents

DDC	TSP equivalent command	Description
Crc	<code>channel.close(channelList)</code>	Closes a channel
Nrc	<code>channel.open(channelList)</code>	Opens a channel
Hn	<code>display.sendkey()</code>	Emulates a keypress
P0	<code>channel.open("allslots")</code>	Opens all channels on all slots
R0	<code>reset()</code>	Restore factory defaults
U2, 0	<code>channel.getclose("allslots")</code>	Get closed channels on all slots
X	No corresponding command	Execute
J0	No corresponding command	Self-test
U7	<code>digio.readport()</code>	Read value from digital I/O port

Relay setup commands

The following table shows DDC commands that refer to Model 707A and 708A relay setups. The Model 707B or 708B equivalent of relay setups are called scans and patterns.

Scans allow a predetermined set of channels to be closed in a sequential order based on a particular event or combination of events. For more information, see [Scanning and triggering](#) (on page 3-1).

There are some distinct differences between relay setups and scans:

- Scans can only be accessed sequentially. Unlike relay setups, you cannot specify an alternate starting location.
- You can only retrieve scan lists in string format.
- You can only append steps to scans — you cannot insert scan steps before steps that are already in the scan list.
- You cannot remove scan steps from the scan list.

The correlation to scanning holds true for DDC commands In , En , $E0$, $Lbbbb$, Pn , and Qn , while the DDC commands $Z0,n$, $Zn,0$, and Zm,n resemble patterns on the Model 707B or 708B instrument. Patterns allow several channels to be associated by a name, and any operation that occurs to the pattern (such as close or open) happens to each channel in the pattern. For more information, see [Channel patterns](#) (on page 2-99).

Scanning trigger

DDC	TSP equivalent command	Description
En	No corresponding command	Point to present relay setup
$E0$	No corresponding command	Point to stored relay setup
$Lbbbb$	<code>print(scan.list())</code>	Download setups
Pn	No corresponding command	Clear relay setup
Qn	<code>scan.create(" ")</code>	Delete setup
$Z0,n$	<code>channel.pattern.snapshot("patternName")</code>	Copy present relays to n
$Zn,0$	<code>channel.close("patternName")</code>	Copy setup from n to relays
Zm,n	<code>channel.pattern.setimage(pattern1, pattern2)</code>	Copy setup from m to n
In	No corresponding command	Insert blank setup to mem

Disable or enable the scan event trigger

When enabling external triggering on the Model 707A or 708A instruments, there were two possible trigger sources, the external trigger line or a GPIB Get command. The Model 707B or 708B has multiple trigger events that are available to iterate through a scan list. The following table shows that specifying a scan event trigger automatically enables it. For more information, see [Scanning and triggering](#).

Scanning trigger

DDC	TSP equivalent command	Description
$F0$	<code>scan.trigger.channel.stimulus = 0</code>	Disable scanning trigger
$F1$	<code>scan.trigger.channel.stimulus = eventID</code>	Enable scanning trigger

Trigger polarity

The Model 707A or 708A instrument has one external trigger input that was available to accept TTL level trigger inputs to advance the stored relay setup step.

The Model 707B or 708B instrument allows 14 TTL level inputs through the digital I/O connector to act as events to iterate through a scan list. The handling of this command is similar to the F0 and F1 DDC commands shown in the table below, except that you must specify *[N]* as the digital I/O line polarity you wish to set. For more information, see [Scanning and triggering](#) (on page 3-1).

Scan trigger polarity

DDC	TSP equivalent command	Description
A0	<code>digio.trigger[N].mode = digio.TRIG_FALLING</code> <code>scan.trigger.channel.stimulus = digio.trigger[N].EVENT_ID</code>	Select falling edge for scan trigger
A1	<code>digio.trigger[N].mode = digio.TRIG_RISING</code> <code>scan.trigger.channel.stimulus = digio.trigger[N].EVENT_ID</code>	Select rising edge for scan trigger

Matrix ready

The Model 707A or 708A instrument provides a dedicated TTL output for when a channel close or open operation completes. This output includes the settling time of the specific channel (fixed in hardware) added to a programmed settling time that could be specified by the user. These commands are shown in the table below.

Matrix ready polarity

DDC	TSP equivalent command	Description
B0	No corresponding command	Rising edge on matrix ready
B1	No corresponding command	Falling edge on matrix ready

The Model 707B or 708B instrument does not have a dedicated matrix ready line when the DDC command compatibility is disabled. However, you can create a TSP function that can emulate the dedicated hardware matrix ready function onto a digital I/O line. The following examples show you how to create TSP code functions for channel close and channel open, respectively, that will duplicate matrix ready functionality for digital I/O bit 10. After including these functions in your script, a channel close operation is made by calling the function with the channel argument:

```
chanClose("1A01")
```

For more information, see [Digital I/O port](#) (on page 2-7).

Example: Channel close with matrix-ready functionality

<code>function chanClose(chan)</code>	Function accepts valid channels as arguments
<code>digio.writebit(10,0)</code>	Clear the 10th digital I/O bit before the close operation
<code>channel.close(chan)</code>	Close the channels in the argument
<code>digio.writebit(10,1)</code>	Set the 10th digital I/O bit after the close operation
<code>end</code>	End the function

Example: Channel open with matrix-ready functionality

<code>function chanOpen(chan)</code>	Function accepts valid channels as arguments
<code>digio.writebit(10,0)</code>	Clear the 10th digital I/O bit before the open operation
<code>channel.open(chan)</code>	Open the channels in the argument
<code>digio.writebit(10,1)</code>	Set the 10th digital I/O bit after the open operation
<code>end</code>	End the function

Set VFD display

The following commands display a value on the VFD front panel. To return the display to power-on default, send the command:

```
display.screen = display.main
```

The display can also be cleared with the `display.clear()` command. For more information, see display functions and attributes.

VFD display

DDC	TSP equivalent command	Description
Dccccccc	<code>display.screen = display.USER</code>	Set VFD display text
	<code>display.settext()</code>	

Set relay setup data format

The G_n DDC command controls the data format of relay setups that are returned when queried with the U_2, s command. The Model 707B or 708B uses a string format for returning scan list, so data formatting is not used.

Relay setup data format

DDC	TSP equivalent command	Description
G_n	No corresponding command	Data format

GPIO EOI hold off control

The K_n DDC command controls how the GPIO EOI line behaves. The Model 707B or 708B commands are processed from an internal buffer as they are received, so no control is required for the GPIO EOI line.

GPIO EOI hold off

DDC	TSP equivalent command	Description
K_n	No corresponding command	GPIO EOI/hold off control

Service request configuration

These commands control the behavior of the SRQ line. The Model 707B or 708B instrument can configure some of the events that cause the SRQ to assert, but none of those events address Matrix Ready or when the instrument is ready for trigger. For more information, see [Status model](#) (on page C-1).

Service request configuration

DDC	TSP equivalent command	Description
M0	<code>status.request_enable = 0</code>	Disable SRQ
M8	No corresponding command	SRQ on Matrix Ready
M16	No corresponding command	SRQ on Matrix Ready
M32	<code>status.request_enable = status.ERROR_AVAILABLE</code>	SRQ on error

Digital I/O control

The Model 707B or 708B instrument provides increased flexibility, with fourteen digital I/O lines (compared to the Model 707A, which has eight input lines and eight output lines, and the Model 708A, which has sixteen input lines and sixteen output lines). It is also possible to write to the digital I/O port simultaneously or on a per-bit basis. For more information, see the [digio functions and attributes](#) (on page 5-9) section.

Digital I/O control

DDC	TSP equivalent command	Description
Onnn	<code>digio.writeport(data)</code>	Sets states of digital I/O
Db,s	or <code>digio.writebit(bit,data)</code>	

Additional channel settling time

The Model 707A or 708A had the ability to add relay settling time to the system after a channel closes or opens. The Model 707B or 708B improves on the settling time and allows you to apply additional settling delay on a per-channel basis. For more information, see [channel functions and attributes](#) (on page 5-5).

Additional channel settling time

DDC	TSP equivalent command	Description
Sn	<code>channel.setdelay(channelList,value)</code>	Additional settling time

Retrieve card settling time

Although base channel settle time cannot be returned, additional delay time that was specified with the `channel.setdelay(channelList,delay)` command can be returned using:

```
channel.getdelay("allslots")
```

Delays for each channel in csv format are returned. For more information, see [channel functions and attributes](#) (on page 5-5).

Card settling time

DDC	TSP equivalent command	Description
U6	No corresponding command	Send longest settling time

Trigger source control

The DDC commands in the following table control the trigger source for advancing the relay setups. The Model 707B or 708B instrument provides similar functionality, but omits triggering on talk and triggering on X. For more information, see [Scanning and triggering](#) (on page 3-1).

Trigger source control

DDC	TSP equivalent command	Description
T0 or T1	No corresponding command	Trigger on Talk
T2 or T3	<code>scan.trigger.channel.stimulus = trigger.EVENT_ID</code>	Trigger on GPIB Get
T4 or T5	No corresponding command	Trigger on X
T6 or T7	<code>scan.trigger.channel.stimulus = digio.trigger[N].EVENT_ID</code>	Trigger on external trigger

Send machine status word

The DDC U0 command provides information on several areas of the Model 707A or 708A instrument in one long csv form. There is no one command to provide the same functionality as the U0 command, but the series of Model 707B or 708B commands shown in the following table can be used to provide very similar information, without the need to parse a long, unwieldy string.

Send machine status word

DDC	TSP equivalent commands	Description
U0	<code>print(localnode.model)</code> <code>print(scan.trigger.channel.stimulus)</code> <code>print(display.getlastkey())</code> <code>print(status.request_enable)</code> <code>print(digio.readport())</code>	Returns the model number Returns trigger stimulus line Returns the last key pressed Gets SRQ Mask Reads Digital I/O port

Error status word

This series of commands provides a descriptive list of error codes and error messages until no more errors are present. For more information, see [errorqueue functions and attributes](#) (on page 5-10).

Error status word

DDC	TSP equivalent command	Description
U1	<code>count = errorqueue.count</code> <code>for x=count,0,1 do</code> <code> errorcode, message = errorqueue.next()</code> <code> print(errorcode)</code> <code> print(message)</code> <code>end</code>	Get the number of errors in the queue For loop Get next error codes Print errorcode Print error messages End

Relay pointer operations

The DDC commands in the following table use relay setups as the Model 707B or 708B instrument would use patterns.

The U2,n DDC command returns the channels that are closed in a format based on the DDC Gn command. The corresponding command for this operation returns the channels contained in the pattern as a csv list. There is no need to send a pattern to the relays. Operations that are to be done to the channels of a pattern are done on the pattern correctly. For more information, see [channel functions and attributes](#) (on page 5-5).

Relay pointer operations

DDC	TSP equivalent command	Description
U2,n	<code>channel.pattern.getimage(pattern)</code>	Point to stored relay setup
U3	No corresponding command	Send relay step pointer

Number of subordinates

The Model 707B or 708B can expand to other Keithley Instrument's products that have a TSP-Link connector available. When the command listed in the following table is executed, it provides the number of nodes present in the TSP-linked system.

A feature of the Model 707B or 708B is that these remote nodes are not limited to Model 707B or 708B instruments. They can be remotely linked to any Keithley product that supports TSP-Link. For more information, see [tsplink functions and attributes](#) (on page 5-16).

Number of subordinates

DDC	TSP equivalent command	Description
U4	<code>print(tsplink.reset())</code>	Send number of subordinate devices

The following command returns the model of the card in the specified slot of the local instrument, along with a description of the card, the firmware version, and the serial number.

To get information from cards in TSP-Link connected instruments, send:

```
print(node[nodeNumber].slot[slotNumber].idn)
```

Where: *nodeNumber* is the TSP-Link node of the remote system and *slotNumber* is the slot on the instrument to query. For more information, see [slot\[X\] attributes](#) (on page 5-15) and [tsplink functions and attributes](#) (on page 5-16).

Card model number

DDC	TSP equivalent command	Description
U5,n	<code>print(slot[slotNumber].idn)</code>	Send identification of each card in the instrument

Relay test input

The Model 707B or 708B instrument does not use a relay test connector, so this command is not supported.

Relay test input

DDC	TSP equivalent command	Description
U8	No corresponding command	Send relay test input

Connect rules

On the Model 707B or 708B, this command sets the connect rules for the entire instrument. You cannot set this rule on a row-by-row basis. If this feature is required, the high performance of the Model 707B or 708B instrument provides ample time to change the connection rules as needed before channel close or open operations. For more information, see [channel functions and attributes](#) (on page 5-5).

Connect rules

DDC	TSP equivalent command	Description
Vabcdefg	<code>channel.connectrule = channel.MAKE_BEFORE_BREAK</code>	Make-before-break
Wabcdefg	<code>channel.connectrule = channel.BREAK_BEFORE_MAKE</code>	Break-before-make

Termination character sent by instrument

The Model 707B or 708B always sends a linefeed (ASCII 10) as a termination character after any data. This behavior cannot be changed.

Termination characters

DDC	TSP equivalent command	Description
Yn	No corresponding command	Termination character

Instrument programming

In this section:

Fundamentals of scripting for TSP	6-1
Fundamentals of programming for TSP	6-15
Using Test Script Builder (TSB)	6-35
Advanced scripting for TSP	6-36
TSP-Link system and running parallel test scripts....	6-45
TSP-Net	6-55

Fundamentals of scripting for TSP

NOTE

The next few sections of the documentation describe scripting and programming features of the instrument. You do not need to review this information if you do not need to use 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. In the instrument, the Test Script Processor (TSP[®]) processes and runs scripts.

Scripts offer several advantages over sending individual commands from the control computer:

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument performs faster and more efficiently when processing scripts.
- 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.

While 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.

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 to prompt the operator to enter parameters.

Runtime and nonvolatile memory storage of scripts

Scripts are loaded into the runtime environment of the instrument. From there, they can be stored in the nonvolatile memory.

The runtime environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to remember a value as long as the instrument is turned on and the variable has not been assigned a new value. After scripts are loaded into the runtime environment, you can run and manage them from the front panel of the instrument or from a computer.

Nonvolatile memory is where information is stored even when the instrument is turned off. To save a script when the instrument is turned off, you must save it to nonvolatile memory. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

Information in the runtime environment is lost when the instrument is turned off.

Scripts are placed in the runtime environment when:

- The instrument is turned on. All scripts that are saved to nonvolatile memory are copied to the runtime environment when the instrument is turned on.
- They are loaded into the runtime environment.

For detail on the amount of memory available in the runtime environment, see [Memory considerations for the runtime environment](#) (on page 6-44).

NOTE

If you make changes to a script in the runtime environment, the changes are lost when the instrument is turned off. To save the changes, you must save them to nonvolatile memory. See [Working with scripts in nonvolatile memory](#) (on page 6-9).

What can be included in scripts?

Scripts can include combinations of commands, instrument control commands, and Lua code.

Commands tell the instrument to do one thing.

Instrument control commands are commands that are specific to instrument control. They are similar to commands sent to a conventional instrument. Each TSP-enabled instrument has an Instrument Control Library (ICL) specific to that instrument, as described in the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8).

Lua is a scripting language that is described in [Fundamentals of programming for TSP](#) (on page 6-15).

Commands that cannot be used in scripts

While the 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.

You can:

- [Create and load a script](#) (on page 6-4)
- [Run the anonymous script](#) (on page 6-7)
- [Save the anonymous script as a named script](#) (on page 6-9)
- [Run scripts](#) (on page 6-6)

Tools for managing scripts

To manage scripts, you can send messages to the instrument, use your own development tool or program, use the Keithley Test Script Builder Integrated Development Environment (TSB IDE), or use TSB Embedded.

The TSB IDE is a programming tool that is included on the product CD. You can use it to create, modify, debug, and store TSP scripts. For information on using the TSB IDE, see [Using Test Script Builder \(TSB\)](#) (on page 6-35).

TSB Embedded is a tool with a reduced set of features from the complete Keithley TSB IDE. TSB Embedded has both script-building functionality and console functionality (single line ICL commands). It is accessed from an Internet browser.

Create and load a script

When you create a script, you can create it as the anonymous script or as a named script.

Once a script is created and loaded into the instrument, you can execute it either 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 runtime environment. If another anonymous script is loaded into the runtime 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 runtime environment). When a named script is created and loaded into the runtime 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 send 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 other variables that reference it.
- Sending revised scripts with different names will not remove previously sent scripts.
- Named scripts are stored in the runtime environment. Therefore, they are lost when the instrument is turned off unless you also save them to nonvolatile memory. See [Working with scripts in nonvolatile memory](#) (on page 6-9).

Create a script by sending commands over the remote interface

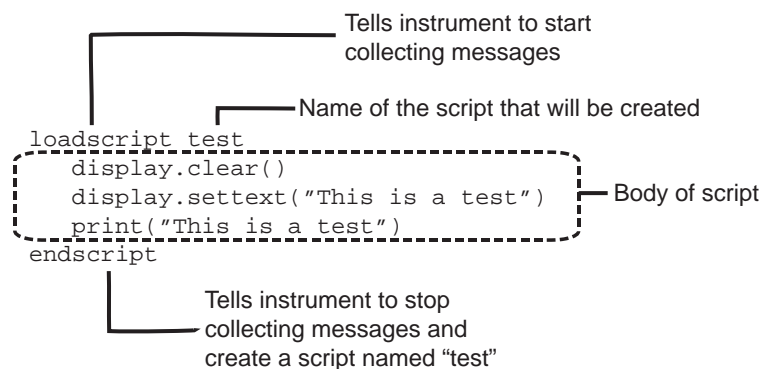
You can send commands over the remote interface instead of using TSB Embedded. To create a script over the remote interface, you can use the `loadscript`, `loadandrunscript`, and `endscript` commands.

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 messages. It compiles the messages into one group of commands. This group of commands is loaded into the runtime environment.

The following figure shows an example of how to load and create 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” will be displayed on the instrument and sent to the computer.

Figure 79: Loadscript and endscript example



To create 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 (see [Lua reserved words](#) (on page 6-16)).
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 6-6).

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`. For information on using TSB Embedded, select the Help button on a web page or the Help option from the navigational pane on the left hand side.

You can create a script in from the instrument web page with TSB Embedded. When you save the script in TSB Embedded, it is loaded into the runtime environment and saved in the nonvolatile memory of the instrument.

To create a script using TSB Embedded:

1. In the TSP Script box, enter the name of the script.
2. In the input area, enter the sequence of commands to be included in the script. Line numbers are automatically assigned.
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 2-106).

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 runtime 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

Any named script that is in the runtime environment can be run 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

Code	Output
<code>test3()</code>	If the script <code>test3</code> is loaded into the runtime environment, the instrument executes <code>test3</code> .

Scripts that run automatically

The autorun scripts and the autoexec script run automatically when the instrument is turned on.

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.

To set a script to run automatically, send a command containing the script name with `.autorun` appended to it, set the autorun attribute to `"yes"`, and then type and send the save command:

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

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

To disable autorun, set the attribute to `"no"`.

NOTE

The `.save()` command saves the script to nonvolatile memory. This saves the change when the instrument is turned off. See [Save a user script to nonvolatile memory](#) (on page 6-9) for more detail.

Example**Code**

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

Output

Assume a script named "test5" is in the runtime 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 6-4).

If you use the create configuration script feature, you can assign the configuration script to be the autoexec script. See [Save the present configuration](#) (on page 2-106) for more information.

NOTE

You need to save the autoexec script to nonvolatile memory to save the script when the instrument is turned off. See [Save a user script to nonvolatile memory](#) (on page 6-9) for more detail.

Example: With loadscript command**Code**

```
loadscript autoexec
display.clear()
display.settext('Hello from autoexec')
endscript
autoexec.save()
```

Notes

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: TSB Embedded**Code**

```
display.clear()
display.settext('Hello from autoexec')
```

Notes

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 a named script and save it to nonvolatile memory:

1. To name the script, send the following command:

```
script.anonymous.name = "myTest"
```

where *myTest* is the name of the script

2. Send the command `script.anonymous.save()` to save *myTest* to nonvolatile memory.

Working with scripts in nonvolatile memory

The previous section, [Fundamentals of scripting for TSP](#) (on page 6-1), described working with scripts, primarily in the runtime environment.

Scripts can also be stored in nonvolatile memory. Information in nonvolatile memory is stored even when the instrument is turned off. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

The runtime environment and nonvolatile memory are separate storage areas in the instrument. The runtime environment is wiped clean 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 runtime environment.

This section describes how to work with the scripts in nonvolatile memory, including how to:

- [Save a user script to nonvolatile memory](#) (on page 6-9)
- [Retrieve a user script](#) (on page 6-10)
- [Restore a script to the runtime environment](#) (on page 6-41)
- [Delete user scripts](#) (on page 6-13)

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 6-4)).
2. Do one of the following:
 - Send the command `myScript.save()`, where *myScript* is the name of the script.
 - In TSB Embedded, click **Save Script**.

Example: Save a user script

Code	Notes
<pre>test1.save()</pre>	Assume a script named "test1" has been created and loaded. <code>test1</code> is saved into nonvolatile memory.

Retrieve a user script

You can retrieve the source code of a user script that is stored in nonvolatile memory.

There are several ways to retrieve the source code for the user script:

- Line by line: Use `scriptVar.list()` to retrieve the source code line by line.
- Entire script: Use the `print(scriptVar.source)` command to retrieve the script source code as a single string.
- Use TSB Embedded.

After retrieving a script, see [Create and load a script](#) (on page 6-4) for information on re-creating the script and loading it back into the instrument.

NOTE

To get a list of scripts that are in nonvolatile memory, see [script.user.catalog\(\)](#) (on page 7-163).

Retrieve source code line by line

To retrieve the source code line by line, use the `scriptVar.list()` command. The source code for this method 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 line by line, 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

Code	Notes
<pre>test7.list()</pre>	Retrieve the source of a script named "test7".

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

Code	Notes
<pre>print(test1.source)</pre>	Retrieve the source of a script named "test7".

If you are using TSB Embedded to retrieve a script:

1. In TSB Embedded, from the User Scripts list, select the script you want to edit. The contents of the script are displayed.
2. If you want to save the script to a new name, change the name in the TSP Script box and click **Save Script**.
3. Edit the content of the script as needed.
4. Click **Save Script**.
5. For the overwrite message, click **OK**.

Script example: Retrieve the content of scripts

This set of examples:

- Creates a script that generates an error.
- Retrieves the source of the script using `scriptVar.source` (on page 7-167).
- Retrieves the source of the script using `scriptVar.list()` (on page 7-164).

Example: Create a script named `scriptVarTest`

Code	Description
<pre>errorqueue.clear() closedChannels = channel.getclose("1F10") if closedChannels == nil then channel.setforbidden("1F10") else channel.open("1F10") channel.setforbidden("1F10") end channel.close("1F10") print(errorqueue.next())</pre>	<p>Clears existing errors.</p> <p>Determine if crosspoint 1F10 is closed.</p> <p>If 1F10 is currently open, configure it as a forbidden channel.</p> <p>If 1F10 is currently closed, open the crosspoint and set it as a forbidden channel.</p> <p>Attempt to close 1F10, which should generate an error.</p> <p>Check the error queue and print the next error.</p>

Example: Retrieve the content of the script with scriptVar.source**Code**

```
print(scriptVarTest.source)

errorqueue.clear()
closedChannels = channel.getclose("1F10")
if closedChannels == nil then
    channel.setforbidden("1F10")
else
    channel.open("1F10")
    channel.setforbidden("1F10")
end
channel.close("1F10")
print(errorqueue.next())
```

Description

Request the source of *scriptVarTest*. The instrument returns the following. Note that the `loadscript` and `endscript` commands are not included

Example: Retrieve the content of the script with scriptVar.list**Code**

```
scriptVarTest.list()
loadscript scriptVarTest
errorqueue.clear()
closedChannels =
    channel.getclose("1F10")
if closedChannels == nil then
    channel.setforbidden("1F10")
else
    channel.open("1F10")
    channel.setforbidden("1F10")
end
channel.close("1F10")
print(errorqueue.next())
endscript
```

Description

Request a listing of the source of *scriptVarTest*. The instrument returns the following. Note that the `loadscript` and `endscript` commands are included.

Delete user scripts

NOTE

These steps remove a script from nonvolatile memory. To completely remove a script from the system, there are additional steps you must take. See [Delete user scripts from the instrument](#) (on page 6-43).

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.

If you are using TSB Embedded, to delete a script from nonvolatile memory:

1. In TSB Embedded, select the script from the User Scripts list.
2. Click **Delete**. There is no confirmation message.

Example

Code

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

Output

Delete a user script named "test8" from nonvolatile memory.

Programming examples

Close channel script

This script closes matrix crosspoints to prepare for actions on other instruments and devices connected to the switch matrix, then opens closed matrix crosspoints.

NOTE

If you are using TSB Embedded, do not include the `loadscript` and `endscript` commands.

Example

Code

```
loadscript closeChannels
reset()
errorqueue.clear()
channel.exclusiveclose('1C01,1D02')

channel.open('allslots')
endscript
```

Description

Resets the instrument to its default state.

Clears the error queue.

Closes matrix crosspoints C01 and D02 on the card in slot 1.

At this point in your code, send commands to perform actions on other instruments and devices connected to the switch matrix.

Opens any closed matrix crosspoints in all slots.

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.

```
-- Clear the display
display.clear()

-- Prompt user for digital I/O line on which to output trigger
myDigioLine = display.menu(
    "Select digital output trigger line", "1 2 3 4 5 6 7 8 9 10")

-- Convert user input to a number
intMyDigioLine = tonumber(myDigioLine)

-- Prompt user for digital output trigger mode
myDigioEdge = display.menu(
    "Select digital output trigger edge mode", "Rising Falling")
if myDigioEdge == "Rising" then
    edgeMode = 2
else
    edgeMode = 1
end

-- Prompt user for output trigger pulsewidth
myPulseWidth = display.prompt(
    "000.0", "us", "Enter trigger pulsewidth", 10, 1, 100)
digio.trigger[intMyDigioLine].mode = edgeMode

digio.trigger[intMyDigioLine].pulsewidth = myPulseWidth

digio.trigger[intMyDigioLine].assert()
display.setcursor(1, 1)

-- Alert the user through the display that the output trigger
  has occurred.
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. Program statements control script execution and provide tools such as variables, functions, branching, and loop control.

TSP[®] is a Lua interpreter. In TSP, the Lua programming language has been extended with the Keithley Instrument Control Library (ICL), which adds instrument control commands to the standard Lua 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) <http://www.lua.org>. Another source of useful information is the [Lua users group](http://lua-users.org) <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.

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 front-panel LOCAL key. display.locallockout = display.lock
Long comment	--[[]]	Use when the comment text is long enough that it wraps to additional lines. The double left brackets signal the beginning of the multi-line comment, and the double right brackets signal the end of the comment.	--[[Displays a menu with three menu items. If the second menu item is selected, the selection will be given the value Test2.]] selection = display.menu("Sample Menu", "Test1 Test2 Test3") print(selection)

Lua reserved words

You cannot use the following words for function or variable names.

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

In addition to the Lua reserved words, you also cannot use command group names as variable names. Doing so will result in the loss of use of the command group. For example, if you send the command `digio = 5`, you cannot access the `digio.*` ICL commands until you cycle the power to the instrument. These groups include:

beeper	gpib
bit	lan
channel	scan
digio	slot
display	status
eventlog	timer
errorqueue	trigger
format	tsplink

Values and variable types

In Lua, you use variables to store values in the runtime environment for later use.

Lua is a dynamically typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been used.

NOTE

Do not create variable names that are the same as the base names of Model 707B or 708B Instrument Control Library (ICL) commands. Doing so will result in the loss of use of those commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until the power to the instrument is turned off and then back on.

Variables can be one of the following types.

Variable types and values

Variable type	Value	Notes
<code>nil</code>	not declared	Nil is the type of the value <code>nil</code> , whose main property is to be different from any other value; usually it represents the absence of a useful value.
<code>boolean</code>	<code>true</code> or <code>false</code>	Boolean is the type of the values <code>false</code> and <code>true</code> . In Lua, both <code>nil</code> and <code>false</code> make a condition <code>false</code> ; any other value makes it <code>true</code> .
<code>number</code>	number	All numbers are real numbers; there is no distinction between integers and floating-point numbers.
<code>string</code>	sequence of words or characters	
<code>function</code>	a block of code	Functions can carry out a task or compute and return values.
<code>table</code>	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.

Example: Nil

Code	Output
<pre>x = nil print(x, type(x))</pre>	<pre>nil nil</pre>

Example: Boolean

Code	Output
<pre>y = false print(y, type(y))</pre>	<pre>false boolean</pre>

Example: String and number

Code	Output
<pre>x = "123" print(x, type(x)) x = x + 7 print(x, type(x))</pre>	<pre>123 string Adding a number to x forces its type to number 1.3000000e+02 number</pre>

Example: Function

Code	Output
<pre>function add_two(parameter1, parameter2) return parameter1 + parameter2 end print(add_two(3, 4), type(add_two))</pre>	<pre>7.0000000e+00 function</pre>

Example: Table

Code	Notes and Output
<pre>atable = {1, 2, 3, 4} print(atable, type(atable)) print(atable[1]) print(atable[4])</pre>	<p>Defines a table with four numeric elements. Note that the "table" value (shown here as a096cd30) will vary.</p> <pre>table: a096cd30 table 1.0000000e+00 4.0000000e+00</pre>

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

Functions

Lua makes it simple to group commands and statements using the `function` keyword. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple parameters.

Functions can be used to form expressions that calculate and return a value; they also can 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 a global variable in the runtime environment. Like all global variables, the functions persist until they are removed from the runtime environment, are overwritten, or the instrument is turned off.

Create functions using the function keyword

Functions are created with a message or Lua code in the form:

```
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function
- *parameterX*: Parameter values. You can have multiple parameters. Change the value defined by *X* for each parameter and use a comma to separate the values.
- *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.

Example 1

Code	Notes and output
<pre>function add_two(parameter1, parameter2) return parameter1 + parameter2 end</pre>	Creates a variable named <code>add_two</code> that has a variable type of function.
<pre>print(add_two(3, 4))</pre>	7.000000000e+00

Example 2

Code	Notes and output
<pre>add_three = function(parameter1, parameter2, parameter3) return parameter1 + parameter2 + parameter3 end print(add_three(3, 4, 5))</pre>	Creates a variable named <code>add_three</code> that has a variable type of function.
	1.200000000e+01

Example 3

Code	Notes and output
<pre>function sum_diff_ratio(parameter1, parameter2) psum = parameter1 + parameter2 pdif = parameter1 - parameter2 prat = parameter1 / parameter2 return psum, pdif, prat end sum, diff, ratio = sum_diff_ratio(2, 3) print(sum) print(diff) print(ratio)</pre>	Returns multiple parameters (sum, difference, and ratio of the two numbers passed to it).
	5.000000000e+00 -1.000000000e+00 6.666666667e-01

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. See [Create a script by sending commands over the remote interface](#) (on page 6-5).

Steps to create a function using a script

Steps	Example
1. In TSB Embedded, enter a name into the TSP Script box	<code>MakeMyFunction</code>
2. Enter the function as the body of the script	This example concatenates two strings: <pre>MyFunction = function (who) print("Hello " .. who) end</pre>
3. Click Save Script	<code>MakeMyFunction</code> now exists on the instrument in a global variable with the same name as the script (<code>MakeMyFunction</code>). 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	<code>MakeMyFunction()</code> This instructs the instrument to run the script, which creates the <code>MyFunction</code> global variable (which is also a function).
5. Run the new function with a value	<code>MyFunction("world")</code> The response message is: Hello world

Group commands using the function keyword

The following script contains Instrument Control Library (ICL) 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	User script created in user's own program
<pre>function myDisplay(name) display.clear() display.setText(name .. "\$N is here!") end</pre>	<pre>loadscript function myDisplay(name) display.clear() display.setText(name.." \$N is here!") end endscript</pre>

Operators

Lua variables and constants can be compared and manipulated 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 this value 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.

Example

Code	Output
<code>print(10 or errorqueue.next())</code>	1.0000000e+01
<code>print(nil or "a")</code>	a
<code>print(nil and 10)</code>	nil
<code>print(false and errorqueue.next())</code>	false
<code>print(false and nil)</code>	false
<code>print(false or nil)</code>	nil
<code>print(10 and 20)</code>	2.0000000e+01

String concatenation

String operators

Operator	Description
..	Concatenates two strings. If both operands are strings or number, they are converted to strings according to Lua coercion rules, as follows: <ul style="list-style-type: none"> Any arithmetic operation applied to a string tries to convert that string to a number, following the usual rules. Whenever a number is used where a string is expected, the number is converted to a string, in a reasonable format.

Example: Concatenation

Code	Output
<pre>print(2 .. 3) print("Hello " .. "World")</pre>	<pre>23 Hello World</pre>

Operator precedence

Operator precedence in Lua follows the order below (from higher to lower priority):

Precedence	Operator
Highest	^ (exponentiation)
.	not, - (unary)
.	*, /
.	+, -
.	.. (concatenation)
.	<, >, <=, >=, ~=, ==
.	and
Lowest	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 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

Code	Output
<pre>if 0 then print("Zero is true!") else print("Zero is false.") end</pre>	Zero is true!

Example: Comparison

Code	Output
<pre>x = 1 y = 2 if x and y then print("Both x and y are true") end</pre>	Both x and y are true

Example: If and else

Code	Output
<pre>x = 2 if not x then print("This is from the if block") else print("This is from the else block") end</pre>	This is from the else block

Example: Else and elseif**Code**

```
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 less 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 less 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 6-24) 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

Example: While**Code**

```
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
```

Notes and output

This loop will exit when
list[element] = nil.

```
Count list elements on
  numeric index:
1.000000000e+00   One
2.000000000e+00   Two
3.000000000e+00   Three
4.000000000e+00   Four
5.000000000e+00   Five
6.000000000e+00   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

Example: Repeat until**Code**

```
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]
```

Notes and output

```
Count elements in list using
  repeat:
1.000000000e+00   One
2.000000000e+00   Two
3.000000000e+00   Three
4.000000000e+00   Four
5.000000000e+00   Five
6.000000000e+00   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.

Example: Numeric for**Code**

```
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
```

Notes and output

The numeric `for` loop repeats a block of code while a control variable runs through an arithmetic progression.

```
Counting from one to three:
1.000000000e+00  One
2.000000000e+00  Two
3.000000000e+00  Three
Counting from one to four, in steps of two:
1.000000000e+00  One
3.000000000e+00  Three
```

Example: Generic for**Code**

```
days = {"Sunday",
         "Monday",   "Tuesday",
         "Wednesday", "Thursday",
         "Friday",   "Saturday"}

for i, v in ipairs(days) do
    print(days[i], i, v)
end
```

Notes and output

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      1.000000000e+00  Sunday
Monday      2.000000000e+00  Monday
Tuesday     3.000000000e+00  Tuesday
Wednesday   4.000000000e+00  Wednesday
Thursday    5.000000000e+00  Thursday
Friday      6.000000000e+00  Friday
Saturday    7.000000000e+00  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 really necessary to return or `break` in the middle of a block, then an explicit inner block can be used.

Example: Break with while statement

Code	Notes
<pre> 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 i == 0 then print("Break value not found") end </pre>	<p>This example defines a break value (<code>breakValue</code>) so that the <code>break</code> statement is used to exit the <code>while</code> loop before the value of <code>k</code> reaches 0.</p> <p>Output: Going to break and i = 3.0000000e+00</p>

Example: Break with while statement enclosed by comment delimiters

Code	Notes
<pre> 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 </pre>	<p>This example defines a break value (<code>breakValue</code>), 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 <code>while</code> loop.</p> <p>Output: Break value not found</p>

Example: Break with infinite loop

Code	Notes
<pre> a, b = 0, 1 while true do print(a,b) a, b = b, a + b if a > 500 then break end end end </pre>	<p>This example uses a <code>break</code> statement that causes the <code>while</code> loop to exit if the value of <code>a</code> becomes greater than 500.</p> <p>Output:</p> <pre> 0.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00 2.000000e+00 2.000000e+00 3.000000e+00 3.000000e+00 5.000000e+00 5.000000e+00 8.000000e+00 8.000000e+00 1.300000e+01 1.300000e+01 2.100000e+01 2.100000e+01 3.400000e+01 3.400000e+01 5.500000e+01 5.500000e+01 8.900000e+01 8.900000e+01 1.440000e+02 1.440000e+02 2.330000e+02 2.330000e+02 3.770000e+02 3.770000e+02 6.100000e+02 </pre>

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"]`.

Example: Loop array

Code	Notes and output
<pre> atable = {1, 2, 3, 4} i = 1 while atable[i] do print(atable[i]) i = i + 1 end </pre>	<p>Defines a table with four numeric elements.</p> <p>Loops through the array and prints each element.</p> <p>The Boolean value of <code>atable[index]</code> evaluates to <code>true</code> if there is an element at that index. If there is no element at that index, <code>nil</code> is returned (<code>nil</code> is considered to be <code>false</code>).</p> <p>Output:</p> <pre> 1.000000e+00 2.000000e+00 3.000000e+00 4.000000e+00 </pre>

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 more. TSP instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP. These libraries are automatically loaded when the TSP starts and do not need to be managed by the programmer.

The following sections provide information on some of the basic Lua standard libraries. For additional information, see the [Lua website](http://www.lua.org) <http://www.lua.org>.

NOTE

When referring to the Lua website, please be aware that TSP uses Lua 5.0.2.

Base library functions

Base library functions	
Function	Description
<code>collectgarbage()</code> <code>collectgarbage(limit)</code>	Sets the garbage-collection threshold to the given limit (in Kbytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, then Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to 0 (which forces a garbage-collection cycle). See Lua memory management (on page 6-31) for more information.
<code>gcinfo()</code>	Returns the number of kilobytes of dynamic memory that TSP is using and the current garbage collector threshold (also in Kbytes). See Lua memory management (on page 6-31) for more information.
<code>print(e1, e2, ...)</code>	Receives any number of arguments, and generates a response message, using the <code>tostring</code> function to convert them to strings. The output is not formatted. For formatted output, you can use the <code>string.format</code> command (see String library functions (on page 6-31)).
<code>tonumber(x)</code> <code>tonumber(x ,base)</code>	Returns <code>x</code> converted to a number. If <code>x</code> is already a number, or a convertible string, then the number is returned; otherwise, it returns <code>nil</code> . An optional argument specifies the base to interpret the numeral. The base may be any integer between 2 and 36, inclusive. In bases above 10, the letter <code>A</code> (in either upper or lower case) represents 10, <code>B</code> represents 11, and so forth, with <code>Z</code> representing 35. In base 10, the default, the number may have a decimal part, as well as an optional exponent. In other bases, only unsigned integers are accepted.
<code>tostring(x)</code>	Receives an argument of any type and converts it to a string in a reasonable format.
<code>type(v)</code>	The possible results of this function are "nil" (a string, not the value nil), number, string, Boolean, table, function, thread, and userdata.

Lua does automatic memory management, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua manages memory automatically by occasionally running 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
<pre>string.byte(s) string.byte(s, i) string.byte(s, i, j)</pre>	<p>Returns the internal numeric codes of the characters <code>s[i]</code>, <code>s[i+1]</code>, ..., <code>s[j]</code>. The default value for <code>i</code> is 1; the default value for <code>j</code> is <code>i</code>.</p> <p>Note that numeric codes are not necessarily portable across platforms.</p>
<pre>string.char(...)</pre>	<p>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.</p> <p>Note that numeric codes are not necessarily portable across platforms.</p>
<pre>string.format(formatstring, ...)</pre>	<p>Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the <code>printf</code> family of standard C functions. The only differences are that the modifiers <code>*</code>, <code>l</code>, <code>L</code>, <code>n</code>, <code>p</code>, and <code>h</code> are not supported and there is an extra option, <code>q</code>. The <code>q</code> option formats a string in a form suitable to be safely read back by the Lua interpreter: the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.</p> <p>For instance, the call:</p> <pre>string.format('%q', 'a string with "quotes" and \n new line')</pre> <p>will produce the string:</p> <pre>"a string with \"quotes\" and \ new line"</pre> <p>The options <code>c</code>, <code>d</code>, <code>E</code>, <code>e</code>, <code>f</code>, <code>g</code>, <code>G</code>, <code>i</code>, <code>o</code>, <code>u</code>, <code>X</code>, and <code>x</code> all expect a number as argument. <code>q</code> and <code>s</code> expect a string.</p> <p>This function does not accept string values containing embedded zeros, except as arguments to the <code>q</code> option.</p>
<pre>string.len(s)</pre>	<p>Receives a string and returns its length. The empty string <code>" "</code> has length 0. Embedded zeros are counted, so <code>"a\000bc\000"</code> has length 5.</p>
<pre>string.lower(s)</pre>	<p>Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged. The definition of what an uppercase letter is depends on the current locale.</p>
<pre>string.rep(s, n)</pre>	<p>Returns a string that is the concatenation of <code>n</code> copies of the string <code>s</code>.</p>
<pre>string.sub(s, i) string.sub(s, i, j)</pre>	<p>Returns the substring of <code>s</code> that starts at <code>i</code> and continues until <code>j</code>; <code>i</code> and <code>j</code> can be negative. If <code>j</code> is absent, it is assumed to be equal to <code>-1</code> (which is the same as the string length). In particular, the call <code>string.sub(s, 1, j)</code> returns a prefix of <code>s</code> with length <code>j</code>, and <code>string.sub(s, -i)</code> returns a suffix of <code>s</code> with length <code>i</code>.</p>
<pre>string.upper(s)</pre>	<p>Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged. The definition of what a lowercase letter is depends on the current locale.</p>

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees.

Math library functions	
Function	Description
<code>math.abs(x)</code>	Returns the absolute value of x .
<code>math.acos(x)</code>	Returns the arc cosine of x .
<code>math.asin(x)</code>	Returns the arc sine of x .
<code>math.atan(x)</code>	Returns the arc tangent of x .
<code>math.atan2(y, x)</code>	Returns the arc tangent of y/x , but uses the signs of both parameters to find the quadrant of the result. (It also handles correctly the case of x being zero.)
<code>math.ceil(x)</code>	Returns the smallest integer larger than or equal to x .
<code>math.cos(x)</code>	Returns the cosine of x .
<code>math.deg(x)</code>	Returns the angle x (given in radians) in degrees.
<code>math.exp(x)</code>	Returns the value e^x .
<code>math.floor(x)</code>	Returns the largest integer smaller than or equal to x .
<code>math.frexp(x)</code>	Returns m and e such that $x = m2^e$, where e is an integer and the absolute value of m is in the range $[0.5, 1]$ (or zero when x is zero).
<code>math.ldexp(x, n)</code>	Returns $m2^e$ (e should be an integer).
<code>math.log(x)</code>	Returns the natural logarithm of x .
<code>math.log10(x)</code>	Returns the base-10 logarithm of x .
<code>math.max(x, ...)</code>	Returns the maximum value among its arguments.
<code>math.min(x, ...)</code>	Returns the minimum value among its arguments.
<code>math.pi</code>	The value of π (3.141592654).
<code>math.pow(x, y)</code>	Returns x^y . (You can also use the expression x^y to compute this value.)
<code>math.rad(x)</code>	Returns the angle x (given in degrees) in radians.
<code>math.random()</code> <code>math.random(m)</code> <code>math.random(m, n)</code>	This function is an interface to the simple pseudo-random generator function <code>rand</code> provided by ANSI C. When called without arguments, returns a uniform pseudo-random real number in the range $[0, 1]$. When called with an integer number m , <code>math.random</code> returns a uniform pseudo-random integer in the range $[1, m]$. When called with two integer numbers m and n , <code>math.random</code> returns a uniform pseudo-random integer in the range $[m, n]$.
<code>math.randomseed(x)</code>	Sets x as the "seed" for the pseudo-random generator: equal seeds produce equal sequences of numbers.
<code>math.sin(x)</code>	Returns the trigonometric sine function of x .
<code>math.sqrt(x)</code>	Returns the square root of x . (You can also use the expression $x^{0.5}$ to compute this value.)
<code>math.tan(x)</code>	Returns the tangent of x .

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 Instrument Control Library (ICL) 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

```
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
```

User script created in user's own program

```
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
```

Example: Create channels with a for loop

User script created in TSB Embedded	User script created in user's own program
<pre>-- Pseudocard assignment necessary only -- if slot is empty -- slot[1].pseudocard = 7072 for k = 1, 8, 2 do chan1 = channel.createspecifier(1, k, 1) chan2 = channel.createspecifier(1, k, 2) print(chan1 .. "," .. chan2) scan.addimagestep(chan1..".."chan 2) end</pre>	<pre>loadscript -- Pseudocard assignment necessary only -- if slot is empty -- slot[1].pseudocard = 7072 for k = 1,8,2 do chan1 = channel.createspecifier(1, k, 1) chan2 = channel.createspecifier(1, k, 2) print(chan1 .. "," .. chan2) scan.addimagestep(chan1 .. "," .. chan2) end endscript</pre>
Output	
<pre>1A01,1A02 1C01,1C02 1E01,1E02 1G01,1G02</pre>	

Using Test Script Builder (TSB)

Installing the TSB software

To install the TSB software:

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

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

Project Navigator

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

Script Editor

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

Programming interaction

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

Advanced scripting for TSP

Global variables and the `script.user.scripts` table

When working with script commands, it can be helpful to understand how scripts are handled in the instrument.

Scripts are loaded into the runtime environment from nonvolatile memory when you turn the instrument on. They are also added to the runtime environment when you load them into the instrument.

There are several types of scripts in the runtime environment:

- Named scripts
- Unnamed scripts
- The anonymous script

When a named script is loaded into the runtime environment:

- A global variable with the same name is created to reference the script more conveniently.
- An entry for the script is added to the `script.user.scripts` table.

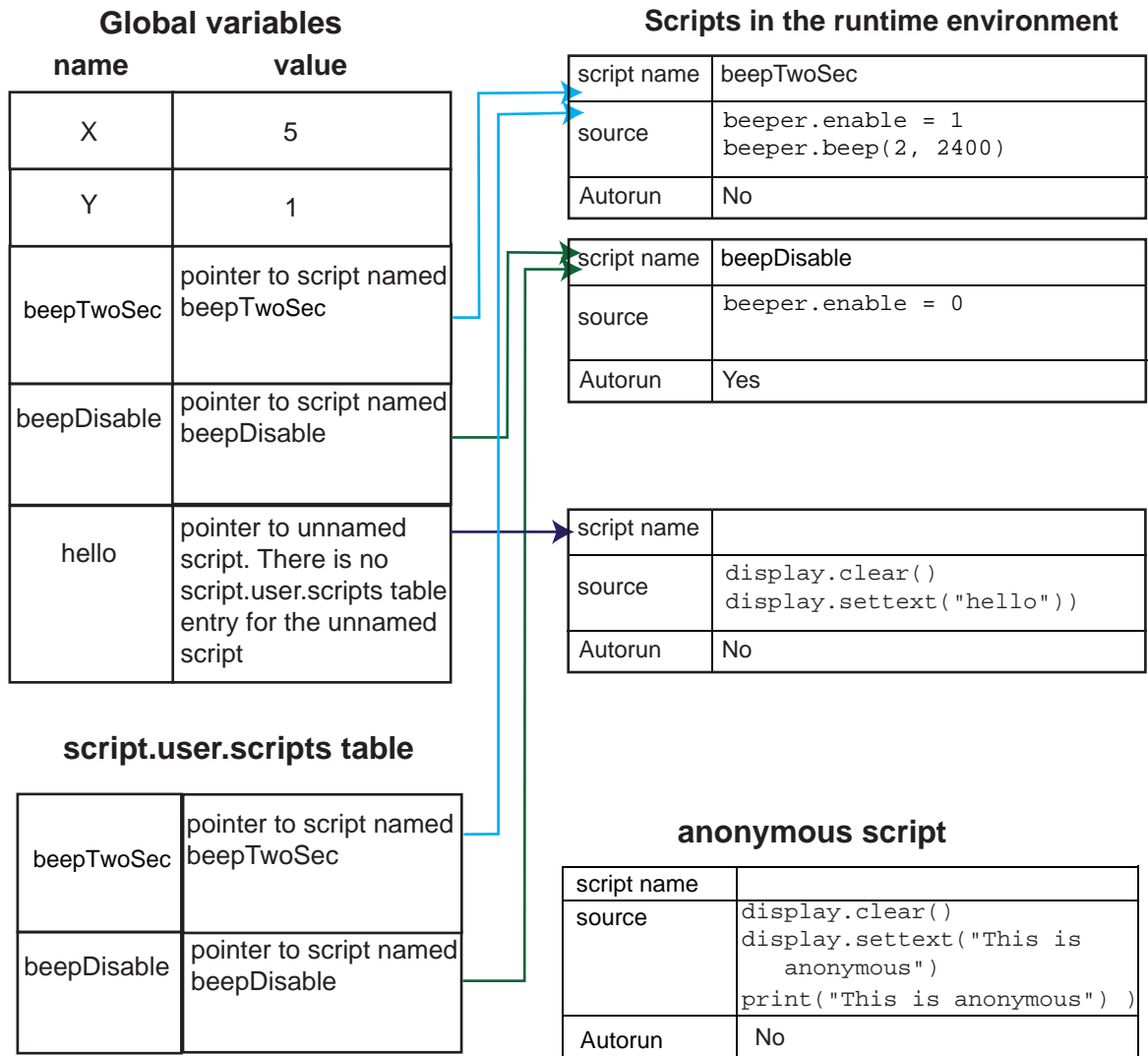
When an unnamed script is loaded, a global variable is added. Nothing is added to the `script.user.scripts` table.

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 runtime 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 80: Global variables and scripts in the runtime environment



Create a script using the script.new command

Use `script.new` to copy an existing script from the local node to a remote node. This can enable parallel script execution.

You can create a script with the `script.new()` function using the command:

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

Where:

- `scriptVar` is the name of the variable that is created when the script is loaded into the runtime environment.
- `code` is the content of the script
- `name` is the name that is added to the `script.user.scripts` table

For example, to set up a two-second beep, you could send the command:

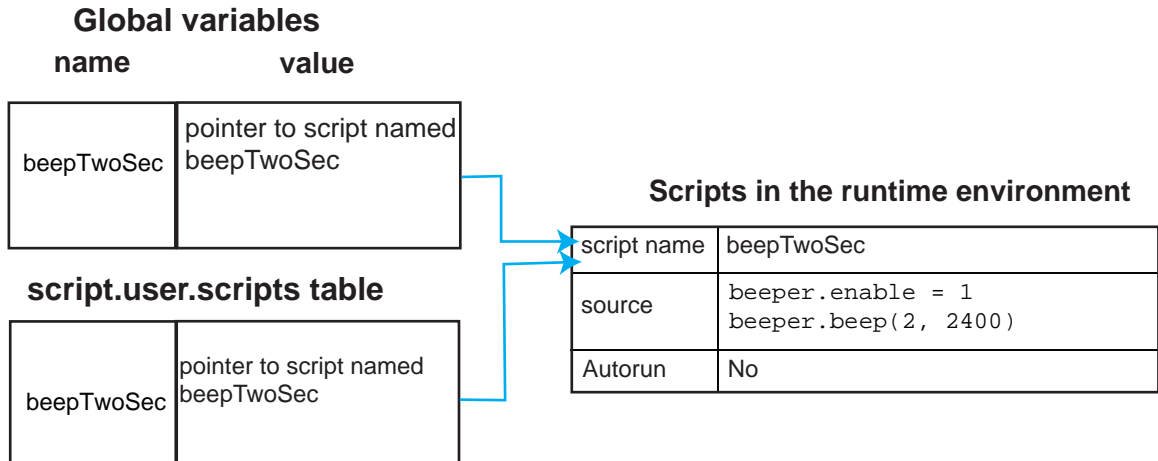
```
beepTwoSec = script.new("beeper.enable = 1 beeper.beep(2, 2400)", "beepTwoSec")
```

To run the new script, you can send the command:

```
beepTwoSec()
```

When you add `beepTwoSec`, the global variable and `script.user.script` table entries are made to the runtime environment as shown in the following figure.

Figure 81: 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 runtime 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, if you sent the command:

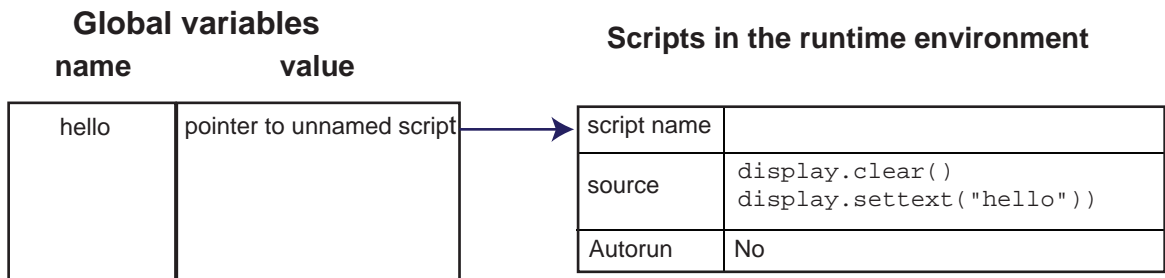
```
hello = script.new('display.clear() display.settext("hello")')
```

A script is created in the runtime environment and a global variable is created that references the script.

To run the script, send the command:

```
hello()
```

Figure 82: Create an unnamed script



Unnamed scripts are also created if you create a new script with the name attribute of a script that is already in the `script.user.scripts` table. In this case, 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 global variable `beepTwoSec` remains in the runtime environment unchanged (it points to the now unnamed script).
- The script that was in the runtime environment as `beepTwoSec` is changed to an unnamed script (the name attribute is set to an empty string).
- A new script named `beepTwoSec` is added to the runtime environment.

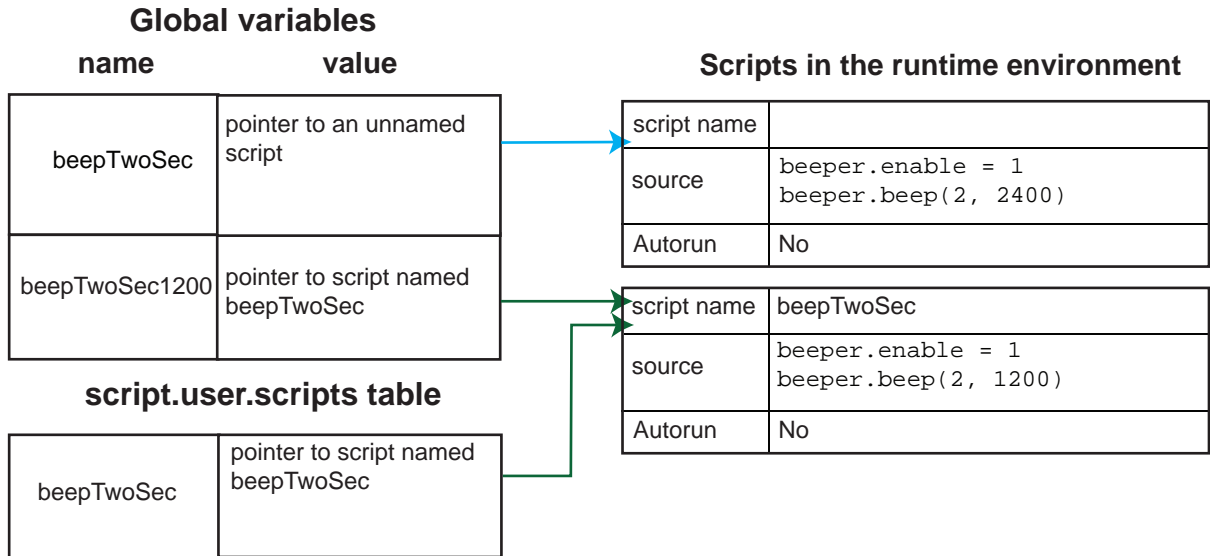
In this example, you can access the new script by sending either of the following commands:

```
beepTwoSec1200()
script.user.scripts.beepTwoSec()
```

To access the unnamed script, you can send the command:

```
beepTwoSec()
```

Figure 83: Change a named script with an unnamed script



Note that the `script.user.scripts` table entry referencing `beepTwoSec` was removed and a new entry for `beepTwoSec` has been added

Restore a script to the runtime environment

You can retrieve a script that was removed from the runtime environment but is still saved in nonvolatile memory.

To restore a script from nonvolatile memory back into the runtime environment:

```
script.restore("scriptName")
```

Where: *scriptName* is the user-defined name of the script to be restored.

Example

Code

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

Output

Restores a user script named "test9" from nonvolatile memory.

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 autoexecute script.

To change the name of a script, use the command:

```
scriptVar.name = "renamedScript"
```

where: *scriptVar* is the global variable name and "*renamedScript*" is 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()
```

beep2sec can be run using the command:

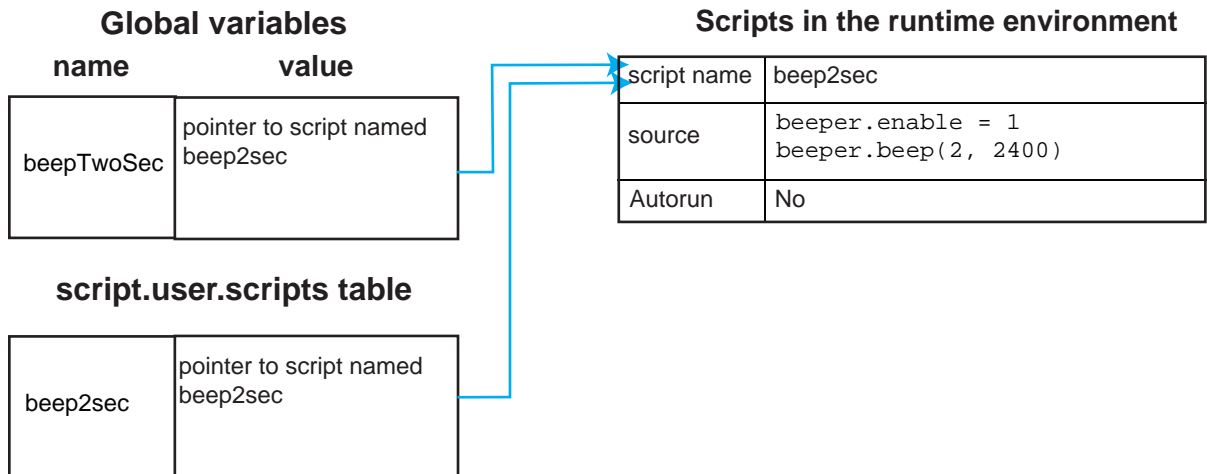
```
script.user.scripts.beep2sec()
```

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 in effect 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 can be found in the `script.user.scripts` table under its new name.

Figure 84: Rename script



Example: Change name of existing script to create the autoexec script

Code

```
test2.name = "autoexec"
test2.save()
```

Notes

Changes name of the script named `test2` to be `autoexec`. 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, "test 2")`
 See `script.new()` (on page 7-160)

Delete user scripts from the instrument

In most cases, you can delete the script using `script.delete()` as described in [Delete user scripts](#) (on page 6-13) and then turning the instrument off and then 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 runtime environment, the `script.user.scripts` table, and nonvolatile memory.

To completely remove a script:

1. **Remove the script from the runtime environment.** Set any global variables that refer to the script to `nil` or assign the variables a different value. For example, to remove the script "beepTwoSec" from the runtime environment, send the following code:
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:

```
beepTwoSec = nil
```

```
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")
```

You can also use TSB Embedded to delete a script from nonvolatile memory. In the TSB Embedded page, select the script from the User Scripts list and click **Delete**.

Memory considerations for the runtime environment

The runtime environment has a fixed amount of memory for storing user scripts, channel patterns, and other runtime information.

You can check the amount of memory in the instrument with 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`, 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.

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`, where `69.14` is the percentage of memory used in the instrument, `0.16` is the percentage used for script storage, and `12.74` is the percentage used for channel pattern storage.

See `memory.available()` (on page 7-135) and `memory.used()` (on page 7-137) for more detail on using these functions.

If the amount of memory used is over 95%, or if you receive out of memory errors, you should reduce the amount of memory that is used.

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 runtime environment when the instrument is turned on. See [Delete user scripts from the instrument](#) (on page 6-43).
- Reduce the number of TSP-Link nodes.
- Delete unneeded channel patterns (this affects only pattern memory, not instrument memory). See [Channel patterns](#) (on page 2-99).
- Delete unneeded global variables from the runtime 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 6-31) for more 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 of optimizing a channel pattern that consists of five channels.

Example

String concatenation lines	Optimized with the table.concat function
<pre>MyPattern = "1A03" MyPattern = MyPattern .. ",1B03" MyPattern = MyPattern .. ",1C03" MyPattern = MyPattern .. ",1D03" MyPattern = MyPattern .. ",1E03" print(MyPattern)</pre>	<pre>MyTable = { } MyTable[1] = "1A03," MyTable[2] = "1B03," MyTable[3] = "1C03," MyTable[4] = "1D03," MyTable[5] = "1E03" MyPattern = table.concat(MyTable) print(MyPattern)</pre>
<p>The output is:</p> <pre>1A03,1B03,1C03,1D03,1E03</pre>	<p>The output is:</p> <pre>1A03,1B03,1C03,1D03,1E03</pre>

**CAUTION**

If the instrument encounters memory allocation errors when the memory used is above 95%, the state of the instrument cannot be guaranteed. After attempting to save off 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 runtime environment. Unsaved scripts and channel patterns will be lost.

TSP-Link system and running parallel test scripts

TSP-Link system

You can use TSP-Link to expand your test system to include up to 64 addressable TSP-Link-enabled instruments (32 instruments at a time). 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.

TSP-Link nodes

Uniquely identify each instrument or enclosure attached to the TSP-Link bus. Identify each instrument using a TSP-Link node number, and each enclosure using a node. Each node must be assigned a unique node number.

In test script programs, nodes look like tables. There is one global table named `node` that contains all the actual nodes that are themselves tables. An individual node is accessed as `node[N]` where `N` is the node number assigned to the node. Each node has certain attributes that can be accessed as elements of its associated table. These are listed as follows:

- `model`: The product model number string of the node.
- `revision`: The product revision string of the node.
- `serialno`: The product serial number string of the node.

There is also an entry for each logical instrument on the node (see [Logical instruments](#) (on page 7-3)).

It is not necessary to know the node number of the node running a script. The variable `localnode` is an alias for the node entry the script is running on. For example, if a script is running on node 5, the global variable `localnode` will be an alias for `node[5]`.

Connect the TSP-Link cable

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.

NOTE

For an example of setting up a TSP-Linked system, see "Working with a Series 2600A" in the Models 707B and 708B User's Manual.

Figure 85: Model 708B rear panel TSP-Link connection

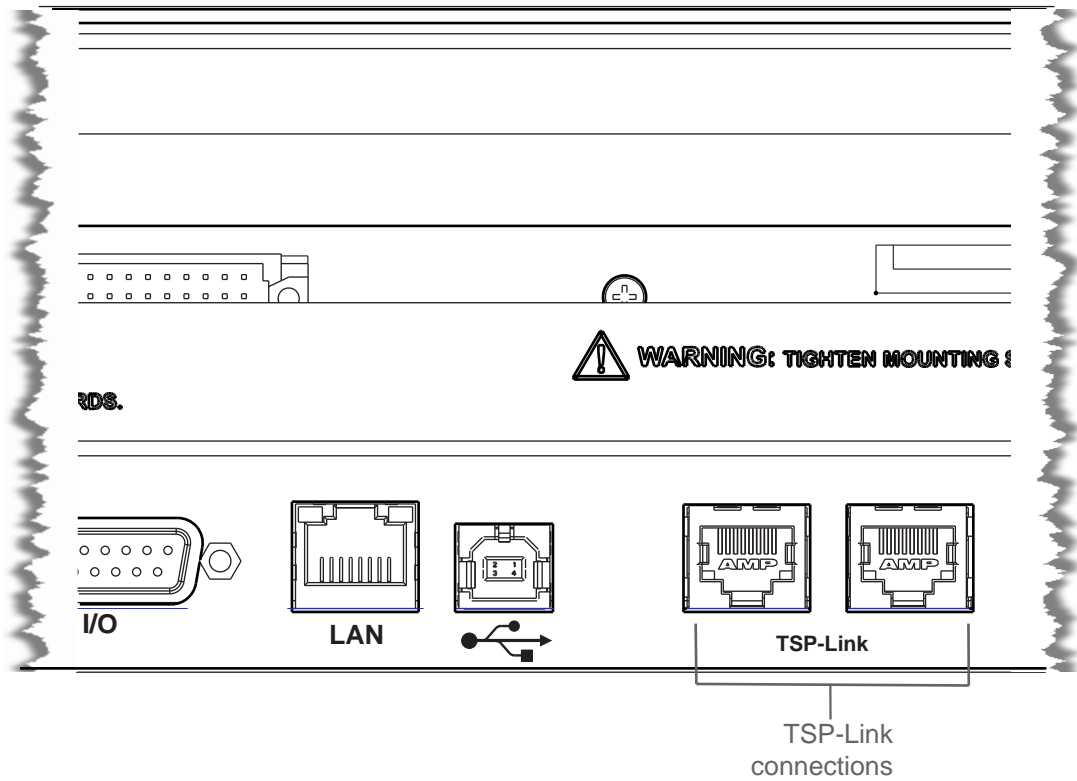
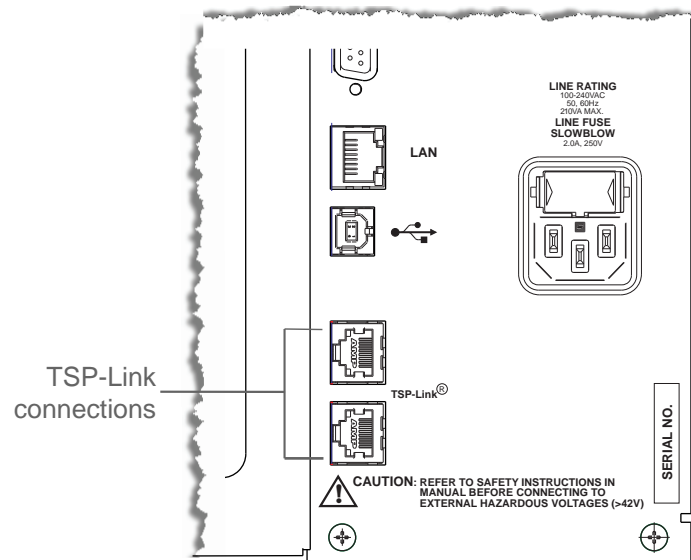


Figure 86: Model 707B rear panel TSP-Link connection



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 Models 707B and 708B instrument is assigned as node 1. The node number for each unit is stored in its nonvolatile memory and will not be lost when the instrument is turned off. You can assign a node number (1 to 64) to a Models 707B and 708B using the front panel or through programming.

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 select the node number.

To assign a node number through programming:

Set the `tsplink.node` attribute of that instrument:

```
tsplink.node = N
```

where: $N = 1$ to 64

The node number of an instrument can be determined by reading the `tsplink.node` attribute as follows:

```
print(tsplink.node)
```

The above `print` command will output the node number. For example, if the node number is 1, the value `1.000000e+00` will be displayed.

Remote programming

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

```
tsplink.node = N
```

Where: $N = 1$ to 64

The node number of an instrument can be determined by reading the `tsplink.node` attribute as follows:

```
print(tsplink.node)
```

The above `print` command will output the node number. For example, if the node number is 1, the value `1.000000e+00` will be displayed.

Resetting the TSP-Link

After all the node numbers are set, you must initialize the system by performing a TSP-Link reset. For initialization to succeed, all units must be powered on when the TSP-Link reset is performed.

NOTE

If you change the system topology after initialization, you must reinitialize the system by performing a TSP-Link reset. Changes that affect the system topology include powering down or rebooting any unit in the system, or rearranging or disconnecting the LAN cable connections between units.

Front panel operation

To reset the TSP-Link network from the front panel:

1. Press the **Menu** key, then select **TSPLINK** and press the **ENTER** key.
2. Turn the navigation wheel to select **RESET** and then press the **ENTER** key.

Remote programming

The commands associated with TSP-Link reset are listed in the following table.

TSP-Link reset commands

Command	Description
<code>tsplink.reset()</code>	Initializes the TSP-Link network.
<code>tsplink.state</code>	Returns "online" if the most recent TSP-Link reset was successful. Returns "offline" if the reset operation failed.

An attempted TSP-Link reset will fail if any of the following conditions are true:

- Two or more instruments in the system have the same node number.
- There are no other instruments connected to the unit performing the reset (only if the expected number of nodes was not provided in the reset call).
- One or more of the units in the system is not powered on.
- 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 is successful, `online` will be returned to indicate that communications with all nodes have been established.

Introduction to TSP advanced features

You can use TSP to run test scripts in parallel on multiple nodes on the TSP-Link network.

You can also use TSP to manage resources allocated to test scripts running in parallel, and to use the data queue to facilitate real-time communication between nodes.

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

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

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

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

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

TSP-Link has three synchronization lines that function similarly to the digital I/O synchronization lines. See [digio functions and attributes](#) (on page 5-9) and [Hardware trigger modes](#) (on page 3-12) for more detailed information.

Using groups to manage nodes on TSP-Link network

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

Master node overview

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

The following list describes the functionality of the master node:

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

Group leader overview

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

The following list describes the functionality of the group leader:

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

Assigning groups

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

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

Note the following:

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

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

Reassigning groups

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

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

Running parallel test scripts

You can issue the `execute` command from the master node to initiate test script and 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 in parallel.

Note the following:

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

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

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

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

NOTE

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

To run `scriptVar` on Node N :

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

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

NOTE

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

To execute 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 6-54) for more information.

Coordinating overlapped operations in remote groups

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

Use the `waitcomplete` command to:

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

For additional information, see [waitcomplete\(\)](#) (on page 7-243).

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

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

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

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

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

Using the data queue for real-time communication

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

The data queue uses the first-in, first-out (FIFO) structure to store data. Nodes running test scripts in parallel can store data in the data queue for real-time communication. Each Models 707B and 708B has an internal data queue. You can access the data queue from any node at any time.

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

To add or retrieve values from the data queue and view the capacity, see the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8).

Copying test scripts across the TSP-Link network

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

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

Note the following:

- Replace *N* with the number of the node that receives a copy of the script
- Replace *scriptName* with the name of the script that you want to copy from the local node

```
-- Adds the source code from scriptName to the data queue.
node[N].dataqueue.add(scriptName.source)
-- Creates a new script on the remote node
-- using the source code from scriptName.
node[N].execute(scriptName.name ..
    "= script.new(dataqueue.next(), [{" .. scriptName.name .. "}]")
```

Removing stale values from the reading buffer

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

Running Lua code remotely can return stale values from the reading buffer to the cached data. 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.

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

Note the following:

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

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

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

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

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

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

-- If you do not use the clearcache command, the data buffer
-- values will never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values will print.
print(rblocal[1])
```

TSP-Net

Overview

TSP-Net™ allows the Model 707B or 708B to control Ethernet-enabled devices directly through its LAN port. This enables the Model 707B or 708B to communicate directly with a device that is not TSP®-enabled without the use of a controlling computer.

TSP-Net Capabilities

For both TSP and non-TSP devices, the TSP-Net library permits the Model 707B or 708B to control a remote device through the LAN port. Using TSP-Net methods, you can transfer string data to and from a remote device, transfer and format data into Lua variables, and clear input buffers. TSP-Net is only accessible using ICL commands from a remote command interface and is not available from the front panel.

You can use TSP-Net to communicate with any Ethernet-enabled device. However, specific TSP-Net commands exist for TSP-enabled devices to allow for support of features unique to TSP. These features include script downloads, reading buffer access, wait completion, and handling of TSP prompts.

Using TSP-Net with TSP-enabled instruments, a Model 707B or 708B can download a script to another TSP-enabled device and have both devices run scripts independently. The Model 707B or 708B can read the data from the remote device and either manipulate the data or send the data to a different remote device on the LAN. You can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the Model 707B or 708B.

Using TSP-Net with any Ethernet-enabled device

NOTE

Refer to the [Instrument Control Library \(ICL\) command reference](#) (on page 7-8) for more details on the commands presented in this section.

To communicate to a remote Ethernet-enabled device from the Model 707B or 708B, perform the following steps:

1. Connect to the remote device through the LAN port. If you are connecting:
 - Directly from the Model 707B or 708B to an Ethernet-enabled device: Use an Ethernet crossover cable.
 - The Model 707B or 708B to any other device on the LAN: Use a straight-through Ethernet cable and a hub.
2. Establish a new connection to a remote device at a specific IP address using `tspnet.connect()`.
3. If the device is not TSP-enabled, you must also provide the port number. If not, the Model 707B or 708B assumes the remote device is TSP-enabled and enables TSP prompts and error handling.

If the Model 707B or 708B 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.

4. 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()` instead, the Model 707B or 708B appends termination characters to all strings sent to the command.
5. Retrieve responses from the remote device using `tspnet.read()`. The Model 707B or 708B 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 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 vs. TSP-Link for communication with TSP-enabled devices

TSP-Link is the preferred communication method when communicating between the Model 707B or 708B and another TSP-enabled instrument. The advantage of using TSP-Link over using TSP-Net include:

- **Error checking:** When connected to a TSP-enabled device, all errors that occur on the remote device are transferred to the error queue of the Model 707B or 708B. The Model 707B or 708B indicates errors from the remote device by prefacing these errors with “Remote Error”.
For example, if the remote device generates error number 4909, the Model 707B or 708B generates the error string “Remote Error: (4909) Reading buffer not found within device.”
- **Digital I/O Triggering:** TSP-Link connections have three TSP synchronization lines that are available to each device on the TSP-Link network. You can use any one of the TSP synchronization lines to perform hardware triggering between devices on the TSP-Link network. Refer to [Hardware trigger modes](#) (on page 3-12) for more details.

These advantages make using TSP-Link to control another TSP-enabled device the best choice for most applications. However, if the distance between the Model 707B or 708B and the TSP-enabled device is longer than 15 feet, use TSP-Net.

To establish a remote TSP-Net connection with a TSP-enabled device, use [`tspnet.connect\(\)`](#) (on page 7-229) without specifying a port number. The Model 707B or 708B enables TSP prompt and error handling for the remote device, which allows you to successfully use the `tspnet.tsp` set of commands to load and run scripts and retrieve reading buffers.

Abort any operation on the remote TSP-enabled device using `abort()`.

Instrument Control Library (ICL) - General device control

The following Instrument Control Library (ICL) commands provide general device control:

[tspnet.clear\(\)](#) (on page 7-228)

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.disconnect\(\)](#) (on page 7-230)

[tspnet.execute\(\)](#) (on page 7-230)

[tspnet.idn\(\)](#) (on page 7-232)

[tspnet.idn\(\)](#) (on page 7-232)

[tspnet.read\(\)](#) (on page 7-232)

[tspnet.readavailable\(\)](#) (on page 7-233)

[tspnet.reset\(\)](#) (on page 7-234)

[tspnet.termination\(\)](#) (on page 7-235)

[tspnet.timeout](#) (on page 7-236)

[tspnet.write\(\)](#) (on page 7-240)

Instrument Control Library — TSP-Enabled device control

The following Instrument Control Library (ICL) commands provide TSP-enabled device control:

[tspnet.tsp.abort\(\)](#) (on page 7-236)

[tspnet.tsp.abortonconnect](#) (on page 7-237)

[tspnet.tsp.rhtablecopy\(\)](#) (on page 7-238)

[tspnet.tsp.runscript\(\)](#) (on page 7-239)

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 via 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 a 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()
```

Command reference

In this section:

Command programming notes.....	7-1
Using the Instrument Control Library documentation .	7-4
Instrument Control Library (ICL) command reference	7-8

Command programming notes

Placeholder characters

The following table lists conventions used in this documentation to emphasize certain parts of the command structure.

Conventions used in this documentation

Convention	What it represents	Examples
Italicized text	Signifies user-defined parameters.	<code>gpib.address = <i>address</i></code> Where: <i>address</i> is an integer (0 to 30) that you specify
Brackets []	<p>Contains placeholder characters for user-defined parameters that must be replaced with a value in the actual code.</p> <p>NOTE Do not send placeholder characters (<i>N</i>, <i>M</i>, <i>X</i>, <i>Y</i>) to the instrument. They are used in this documentation for notational convenience only.</p> <p>The placeholder characters used in this documentation: <i>[N]</i> is typically used with triggers <i>[M]</i> is typically used with trigger blender functions to indicate the stimulus <i>[X]</i> is typically used to indicate the slot <i>[Y]</i> is typically used to indicate a limit number</p>	<p>The function to assert an output trigger is shown as: <code>digio.trigger[N].assert</code> For example, to program the Model 707B or 708B to assert an output trigger on trigger line 5, you would send: <code>digio.trigger[5].assert()</code></p> <p>The attribute return a string containing information about a card in a specified slot is shown as: <code>slot[X].idn</code> For example, to get information about card installed in slot 1, you would send: <code>print(slot[1].idn)</code></p>

Syntax rules

The following table lists syntax requirements to build well-formed instrument control commands.

Syntax rules for Instrument Control Library commands

Syntax rule	Details	Examples
<p>Case sensitivity: Instrument control commands are case sensitive.</p>	Function and attribute names should be in lowercase characters.	An example of the <code>scriptVar.save()</code> function (where <code>test8</code> is the name of the script): <code>test8.save()</code>
For best results, simply match the case shown in the instrument control command descriptions.	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, <code>format.REAL64</code> is the attribute constant and <code>format.data</code> is the attribute command: <code>format.data = format.REAL64</code>
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: <code>digio.writebit(3,0)</code> <code>digio.writebit (3, 0)</code>
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 must complete before commands from other groups can execute: <code>waitcomplete(G)</code> The command below reads the value of the local time zone (no parameters are needed): <code>timezone = localnode.gettimezone()</code>
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: <code>beeper.beeper(0.5, 2400)</code> <code>delay(0.250)</code> <code>beeper.beeper(0.5, 2400)</code>
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 in row 1 columns 1 through 10 on Slot 1: <code>>scan.create("1A01:1A10")</code>

Logical instruments

You would normally refer to all instrumentation in one enclosure or node as a single instrument. In the context of TSP and instrument control libraries, it is useful to think of each individual subdivision in an enclosure, such as a card slot or the channels, as a standalone 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 ICL commands that control a given logical instrument. The logical instruments are:

- beeper
- bit
- channel
- digio
- display
- eventlog
- errorqueue
- format
- gpib
- lan
- scan
- slot
- status
- timer
- trigger
- tsplink

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 ICLs. For example, if you send the command `digio = 5`, you cannot access the `digio.*` ICL commands until you cycle the power to the instrument.

Time and date values

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." There are three time bases:

- **UTC 12:00 am Jan 1, 1970.** Some examples of UTC time are reading buffer base timestamps, 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 Instrument Control Library documentation

The Instrument Control Library documentation contains detailed descriptions of each of the instrument control commands you can use to control your Models 707B and 708B. Each instrument control command description is broken into several standardized subsections. The figure below shows an example of an instrument control command description.

Figure 87: Example instrument control library 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	System reset	Create configuration script	1

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 7-74)

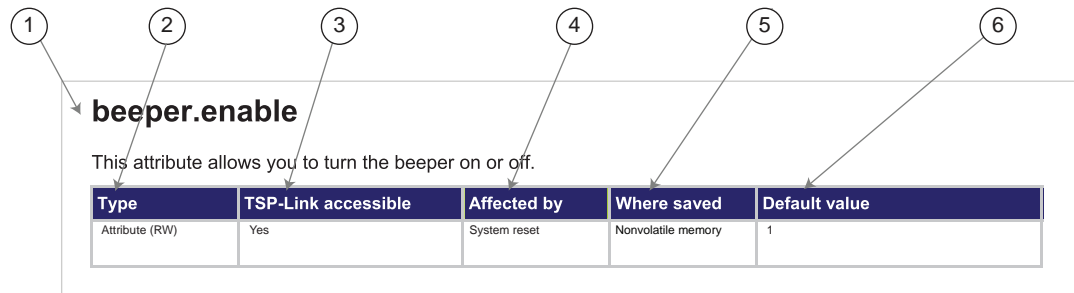
Each instrument control command listing is divided into five major categories of information about the command:

- Command name and standard parameters table
- Usage
- Details
- Example
- Also see

The content of each of these categories is described in the following topics.

Command name and standard parameters summary

The beginning of each ICL 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 88: Command name and standard parameters summary

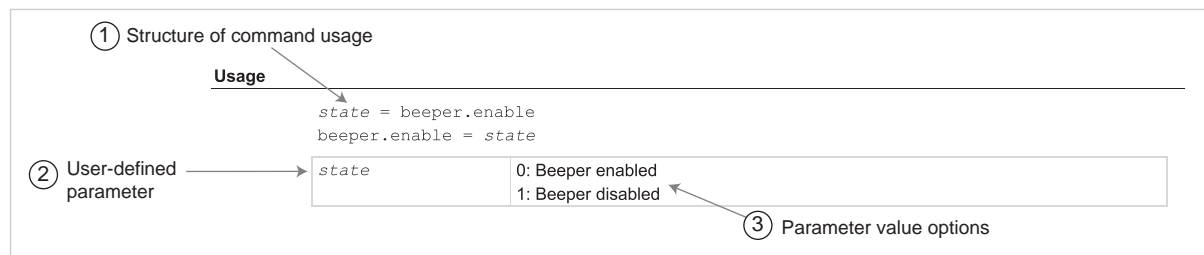
1. **Instrument control command name.** Signals the beginning of the command description and is followed by a brief description of what the command does.
2. **Type of command.** Options are:
 - **Function.** Function-based commands control actions or activities, but are not always directly related to 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.
3. **TSP-Link accessible. Yes or No;** indicates whether or not the command can accessed through a TSP-Link network.
4. **Affected by.** Commands or actions that have a direct effect on the instrument control command.
 - **LAN restore defaults**
 - **Reset:** This command has varied effects depending on how it is used. Reset actions include:
 - Channel reset
 - Digital I/O trigger reset
 - Instrument reset
 - LAN reset
 - LAN restore defaults
 - Local node reset
 - Scan reset
 - Status reset
 - System reset
 - Trigger blender reset
 - Trigger timer reset
 - TSP-Link trigger reset

5. **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.
 - **System settings**
 -
6. **Default value:** Presents the default value or constant for the command. If you read a value, a number is returned. The returned numbers are defined in the Usage or Details sections of the command description.

Command usage

The **Usage** section of the instrument control 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 89: Command usage section



1. **Structure of command usage:** Shows how the parts of the command should be organized.
2. **User-defined 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. For example, `beeper.beep(2, 2400)` generates a two-second, 2400 Hz tone.

NOTE

If there are optional parameters, they must be entered in the order presented in usage. You cannot leave out any parameters that precede the optional parameter, but you can leave off subsequent optional parameters. Optional parameters are shown as separate lines in usage, showing each permutation of the command. For example:

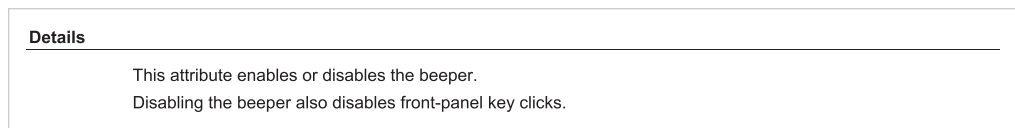
```
text = display.gettext()
text = display.gettext(embellished)
text = display.gettext(embellished, row)
text = display.gettext(embellished, row, columnStart)
text = display.gettext(embellished, row, columnStart, columnEnd)
```

3. Parameter value options: Displayed in order, from left to right, as they are presented in the Usage example.

Command details

This section lists additional information you need to know to successfully use the instrument control command.

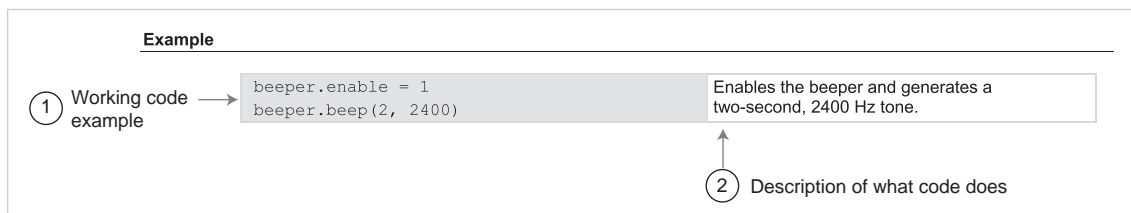
Figure 90: Details section of command listing



Example section

The Example section of the instrument control command description shows some simple examples of how the command can be used.

Figure 91: 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.

Also see: Related commands and information

The **Also see** section of the instrument control command description lists commands that are related to the command being described.

Figure 92: Links to related commands and information

Also see
beeper.beep() (on page 6-21)

Instrument Control Library (ICL) command reference

beeper.beep()

This function generates an audible tone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`beeper.beep(duration, frequency)`

<i>duration</i>	The amount of time to play the tone in seconds; the allowable range is 0.1 s to 100 s
<i>frequency</i>	The frequency of the tone in Hertz (Hz)

Details

The beeper will not sound if it is disabled. It can be disabled or enabled with `beeper.enable`, or through the front-panel Main Menu.

Example

```
beeper.enable = 1
beeper.beep(2, 2400)
```

Enables the beeper and generates a two-second, 2400 Hz tone.

Also see

[beeper.enable](#) (on page 7-9)

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	Reset	Create configuration script	Enabled

Usage

```
state = beeper.enable
beeper.enable = state
```

<i>state</i>	beeper.OFF or 0: Beeper disabled beeper.ON or 1: Beeper enabled
--------------	--------------------------------------------------------------------

Details

Disabling the beeper also disables front panel key clicks.

Example

<pre>beeper.enable = beeper.ON beeper.beep(2, 2400)</pre>	Enables the beeper and generates a two-second, 2400 Hz tone
-----------------------------------------------------------	-------------------------------------------------------------

Also see

beeper.beep()

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)
```

<i>result</i>	Result of the logical AND operation
<i>value1</i>	Operand for the logical AND operation
<i>value2</i>	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: 8.0000000e+00

Also see

[bit.bitor\(\)](#) (on page 7-10)

[bit.bitxor\(\)](#) (on page 7-11)

[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the logical OR operation
<i>value1</i>	Operand for the logical OR operation
<i>value2</i>	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: 1.1000000e+01

Also see

[bit.bitand\(\)](#) (on page 7-9)

[bit.bitxor\(\)](#) (on page 7-11)

[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the logical XOR operation
<i>value1</i>	Operand for the logical XOR operation
<i>value2</i>	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: 3.0000000e+00

Also see

[bit.bitand\(\)](#) (on page 7-9)
[bit.bitor\(\)](#) (on page 7-10)
[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	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

```
myResult = bit.clear(15, 2)
print(myResult)
```

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: 1.3000000e+01

Also see

[bit.get\(\)](#) (on page 7-12)

[bit.set\(\)](#) (on page 7-14)

[bit.test\(\)](#) (on page 7-16)

[bit.toggle\(\)](#) (on page 7-17)

[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> 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 *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

```
myResult = bit.get(10, 4)
print(myResult)
```

The binary equivalent of decimal 10 is 1010. If you get the bit at *index* position 4, the returned decimal value is 8.
Output: 8.0000000e+00

Also see

[bit.clear\(\)](#) (on page 7-11)
[bit.set\(\)](#) (on page 7-14)
[bit.test\(\)](#) (on page 7-16)
[bit.toggle\(\)](#) (on page 7-17)
[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to get (1 to 32)
<i>width</i>	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 *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:
6.0000000e+00

Also see

[bit.get\(\)](#) (on page 7-12)
[bit.set\(\)](#) (on page 7-14)
[bit.setfield\(\)](#) (on page 7-15)
[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> 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 *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

<pre>myResult = bit.set(8, 3) print(myResult)</pre>	<p>The binary equivalent of decimal 8 is 1000. If the bit at <i>index</i> position 3 is set to 1, the returned value is decimal 12 (binary 1100).</p> <p>Output: 1.2000000e+01</p>
-----------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

- [bit.clear\(\)](#) (on page 7-11)
- [bit.get\(\)](#) (on page 7-12)
- [bit.getfield\(\)](#) (on page 7-13)
- [bit.setfield\(\)](#) (on page 7-15)
- [bit.test\(\)](#) (on page 7-16)
- [bit.toggle\(\)](#) (on page 7-17)
- [Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> to set (1 to 32)
<i>width</i>	The number of bits to include in the field (1 to 32)
<i>fieldValue</i>	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 *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

<pre>testResult = bit.setfield(15, 2, 3, 5) print(testResult)</pre>	<p>The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at <i>index</i> position 2, the returned <i>value</i> is decimal 11 (binary 1011).</p> <p>Output: 1.1000000e+01</p>
---------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[bit.get\(\)](#) (on page 7-12)

[bit.set\(\)](#) (on page 7-14)

[bit.getfield\(\)](#) (on page 7-13)

[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	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 *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

```
myResult = bit.test(10, 4)
print(myResult)
```

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.clear\(\)](#) (on page 7-11)

[bit.get\(\)](#) (on page 7-12)

[bit.set\(\)](#) (on page 7-14)

[bit.toggle\(\)](#) (on page 7-17)

[Logical operators](#) (on page 6-22)

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)
```

<i>result</i>	Result of the bit manipulation
<i>value</i>	Specified number
<i>index</i>	One-based bit position within <i>value</i> 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 decimal value is also an integer.

The least significant bit *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

```
myResult = bit.toggle(10, 3)

print(myResult)
```

The binary equivalent of decimal 10 is 1010. Toggling the bit at *index* position 3 returns a decimal value of 14 (binary 1110).
Output: 1.4000000e+01

Also see

- [bit.clear\(\)](#) (on page 7-11)
- [bit.get\(\)](#) (on page 7-12)
- [bit.set\(\)](#) (on page 7-14)
- [bit.test\(\)](#) (on page 7-16)
- [Logical operators](#) (on page 6-22)

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)
```

<i>channelList</i>	String specifying 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 for Model 707B, or 1 for Model 708B)
- Channel ranges or individual channels

This function allows all items contained in the *channelList* parameter to be closed (removes the "forbidden to close" attribute that can be applied to a channel using `channel.setforbidden()`).

Command processing will stop as soon as an error is detected. If an error is found, the channels are not cleared from being forbidden to close.

Example

```
channel.reset("slot1")
channel.setforbidden("1A01:1A05")

channel.clearforbidden("1A02,1A03")
print(channel.getforbidden("slot1"))
```

Reset the channels on slot 1.
Set channels 1A01, 1A02, 1A03, 1A04, and 1A05 to be forbidden to close.
Change 1A02 and 1A03 to be allowed to close.
Retrieve the list of forbidden channels.
Output:
1A01, 1A04, 1A05

Also see

[channel.getforbidden\(\)](#) (on page 7-30)

[channel.setforbidden\(\)](#) (on page 7-45)

channel.close()

This function closes channels and channel patterns 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)
```

<i>channelList</i>	A string that lists the channels and channel patterns to close
--------------------	----------------------------------------------------------------

Details

These closures are appended to the already closed channels (no previously closed channels are opened by this command).

Actions associated with this function include:

- Close the specified items in *channelList*
- Incur the settling time and any user-specified delay

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

Once an error is detected, the command stops processing and no channels are closed.

Example

```
channel.open("allslots")
channel.pattern.setimage("1B02,1B04,1B06",
    "Chans")
channel.close("1A01:1A05, 1C03, Chans")
print(channel.getclose("slot1"))
```

Close a variety of channels, directly and with a channel pattern; note that the output sorts the channels

Output:

```
1A01;1A02;1A03;1A04;1A05;1B02;1B04;
1B06;1C03
```

Also see

- [channel.exclusiveclose\(\)](#) (on page 7-24)
- [channel.exclusiveslotclose\(\)](#) (on page 7-25)
- [channel.getclose\(\)](#) (on page 7-27)
- [channel.open\(\)](#) (on page 7-35)

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	System reset	Create configuration script	channel.BREAK_BEFORE_MAKE

Usage

```
rule = channel.connectrule
channel.connectrule = rule
```

<i>rule</i>	<ul style="list-style-type: none"> 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()`, 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 is 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 7-21)
- [channel.exclusiveclose\(\)](#) (on page 7-24)
- [channel.exclusiveslotclose\(\)](#) (on page 7-25)
- [scan.background\(\)](#) (on page 7-144)
- [scan.execute\(\)](#) (on page 7-147)

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	System reset	Create configuration script	<code>channel.OFF</code>

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 close (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 7-20)

[Switch operation](#) (on page 2-87)

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

```
specifier = channel.createspecifier(slot, row, column)
```

<i>specifier</i>	The formatted string for the channel
<i>slot</i>	The slot number to use (1 to 6)
<i>row</i>	The row number to use (see the matrix card documentation for limits); map row letters to numbers (A = 1, B = 2, and so on)
<i>column</i>	Specifies the column number 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.

Example 1

```
cs = channel.createspecifier(1, 1, 1)
print(cs)
```

Creates a channel descriptor for Row 1, column 1 on the card in Slot 1
Output:
1A01

Example 2

```
count = 0
for row = 1, 8 do
  for col = 1, 12 do
    ch = channel.createspecifier(1, row, col)
    count = count + tonumber(channel.getcount(ch))
  end
end
print("Count is " .. count .. ".")
```

Assuming an 8x12 matrix card in slot 1, this example calculates the sum of the counts on all channels
Output:
Count is 1060656.

Also see

None

channel.exclusiveclose()

This function closes the specified channels and channel patterns so that they are exclusively closed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.exclusiveclose(channelList)
```

<i>channelList</i>	A string listing the channels and channel patterns to exclusively close.
--------------------	--------------------------------------------------------------------------

Details

This command allows you to bundle the closing of channels with opening. Any presently closed channel opens if it is not specified to be closed in the parameter. This guarantees that only the specified items are closed on the slots specified in the parameters list.

Actions associated with this function include:

- Opens previously closed channels if they are no longer specified for closure, and then closes the channels in *channelList*.
- Incur settling and user-specified delay times depending on the connection rules.

If the *channelList* parameter is an empty string or a string of spaces, all channels that are closed are opened. Therefore, when channels are closed, sending `channel.exclusiveclose(" ")` is equivalent to `channel.open(channel.getclose("allslots"))`.

An error is generated if:

- The parameter string contains `slotX`, where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B or `allslots`
- A specified channel or channel pattern is invalid
- A forbidden item is specified

Once an error is detected, the command stops processing. Channels open or close only if no errors are found.

Example

```
channel.exclusiveclose( " " )
channel.close( "1D01,1E12" )
print(channel.getclose( "slot1" ) )

channel.pattern.setimage( "1B02,1B04,1B06", "myChans" )
channel.exclusiveclose( "1A01:1A05, 1C03, myChans" )
print(channel.getclose( "slot1" ) )
```

Open all channels because the `channelList` parameter is empty.

Close 1D01 and 1E12.

Output:

```
1D01;1E12
```

Create the pattern `myChans`.

Exclusively close the channels in `myChans` and additional specified channels. The originally closed channels (1D01 and 1E12) are now open.

Output:

```
1A01;1A02;1A03;1A04;1A05;1B02;1B04;1B06;1C03
```

Also see

[channel.clearforbidden\(\)](#) (on page 7-18)

[channel.close\(\)](#) (on page 7-19)

[channel.connectrule](#) (on page 7-20)

[channel.connectsequential](#) (on page 7-21)

[channel.exclusiveslotclose\(\)](#) (on page 7-25)

[channel.getclose\(\)](#) (on page 7-27)

[channel.getforbidden\(\)](#) (on page 7-30)

[channel.open\(\)](#) (on page 7-35)

[channel.setforbidden\(\)](#) (on page 7-45)

channel.exclusiveslotclose()

This function exclusively closes the specified channels and channel patterns on the defined slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.exclusiveslotclose(channelList)
```

<i>channelList</i>	A string that lists the channels and channel patterns to exclusively close on the card in an associated slot
--------------------	--------------------------------------------------------------------------------------------------------------

Details

This command allows you to bundle the closing of channels with the opening of a channel. Any currently closed channel opens if not specified for closure on the slots defined in the parameter. Using this command guarantees that only the specified channels and channel patterns are closed on the slots associated with the *channelList*.

When this command is sent:

- Closed channels for the defined slots are opened if they are not specified in the *channelList*
- Channels specified by the items in *channelList* are closed
- Any settling times and user-specified delay times are incurred

For example, if row 1, column 1 channels are closed on each of the six slots, specifying a *channelList* parameter of "2A02, 4A04" opens the row 1, column 1 channels (slots 2 and 4 only). Then, the row 1, column 2 channel on slot 2, and the row 1, column 4 channel on slot 4 close. The row 1, column 2 channels remain closed on slots 1, 3, 5, and 6.

An error is generated if:

- The parameter string contains `slotX`, where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B, or `allslots`
- A forbidden item is specified

Once an error is detected, the command stops processing.

Example

```
channel.open("allslots")
channel.close("1A01,2A01,3A01,4A01,5A01,6A01")
channel.exclusiveslotclose("3A03")
print(channel.getclose("allslots"))
```

Open all channels on all slots.
Close row A column 1 channels on all slots.
Open row A column 1 on slot 3 and close row A column 3 on slot 3 without affecting any other slot with closed channels.
Output:
1A01 ; 2A01 ; 3A03 ; 4A01 ; 5A01 ; 6A01

Also see

- [channel.clearforbidden\(\)](#) (on page 7-18)
- [channel.close\(\)](#) (on page 7-19)
- [channel.exclusiveclose\(\)](#) (on page 7-24)
- [channel.getclose\(\)](#) (on page 7-27)
- [channel.getforbidden\(\)](#) (on page 7-30)
- [channel.open\(\)](#) (on page 7-35)
- [channel.setforbidden\(\)](#) (on page 7-45)

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)
```

<i>closed</i>	A string listing the channels that are currently closed
<i>channelList</i>	A string representing the channels that will be queried, including channel patterns

Details

If more than one channel is closed, they are semicolon-delimited in the string. When the *channelList* contains a channel pattern, only the closed channels in that image are returned.

The *channelList* parameter indicates the scope of channels affected, and can include:

- `allslots` or `slotX` (where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B)
- Channel ranges, individual channels, or channel patterns

An error message is generated if an empty parameter string is specified or the scope of the channel list specified contains no valid channels (for example, a channel list equaling "slotX" or "allslots").

If nothing is closed within the specified scope, a `nil` response is returned.

Example 1

```
channelList = "1A01:1H12"
channel.close("1A01")
print(channel.getclose(channelList))
channel.close("1C03")
print(channel.getclose(channelList))
```

For this example, assume there is a card or pseudocard in Slot 1 with no previously closed channels. The output is:

```
1A01
1A01;1C03
```

Example 2

```
channel.close("1B03:1B05")
print(channel.getclose("allslots"))
```

For this example, assume there is a card or pseudocard in Slot 1 with no previously closed channels. The output is:

```
1B03;1B04;1B05
```

Also see

- [channel.close\(\)](#) (on page 7-19)
- [channel.exclusiveclose\(\)](#) (on page 7-24)
- [channel.getstate\(\)](#) (on page 7-34)
- [channel.open\(\)](#) (on page 7-35)

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)
```

<i>counts</i>	A comma-delimited string listing the channel close counts
<i>channelList</i>	A string listing the items to query, which can include: <ul style="list-style-type: none"> • Channels • Channel patterns (channels will be listed in the order in which they are listed in the pattern) • slotX, where X equals 1 to 6 for the Model 707B or 1 for the Model 708B • 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.

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 are listed before slot 2 channels.

The counts reported for the following cards indicate the number of closures since the last power cycle of the card:

- 7072
- 7072-HV
- 7173-50
- 7174A

For all other cards, the number of closures is the closures that have occurred over the lifetime of the card.

If an error is detected, a `nil` value is returned. No partial list of close counts is returned.

Example

<pre>channel.pattern.setimage("1A01,1B02,1C01","Path") PathList = channel.pattern.getimage("Path") print(PathList) print(channel.getcount(PathList)) print(channel.getcount("Path"))</pre>	<p>Gets the close counts for channels in a channel pattern called "Path"</p> <p>Sample output:</p> <pre>1A01,1B02,1C01 11001,11014,11025 11001,11014,11025</pre>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[channel.pattern.getimage\(\)](#) (on page 7-38)

[channel.pattern.setimage\(\)](#) (on page 7-39)

Data retrieval commands

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	System reset Channel reset	Create configuration script	0

Usage

```
delayTimes = channel.getdelay(channelList)
```

<i>delayTimes</i>	A comma-delimited string consisting of the delay times (in seconds) for channels specified in <i>channelList</i> .
<i>channelList</i>	A string listing the channels to query for their delay times.

Details

The *channelList* parameter may contain *slotX* (where X equals 1 to 6 for Model 707B, or 1 for Model 708B) 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

```
print(channel.getdelay("1A07,1B05,1C03"))
```

Get the existing delays for the listed channels.

Output:

```
0.0000000e+00,0.0000000e+00,0.0000000e+00
```

```
channel.setdelay("slot1", 3.1)
```

```
DelayTimes =
```

```
channel.getdelay("1A07,1B05,1C03")
```

```
print(DelayTimes)
```

Set a delay on all channels in slot 1.

Verify that the delay was set for the listed channels.

Output:

```
3.1000000e+00,3.1000000e+00,3.1000000e+00
```

Also see

[channel.setdelay\(\)](#) (on page 7-44)

Data retrieval commands

channel.getforbidden()

This function returns a string listing the channels in the channel list parameter that are forbidden to close.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Permitted to close

Usage

```
forbiddenList = channel.getforbidden(channelList)
```

<i>forbiddenList</i>	Comma-delimited string listing channels that are forbidden to close in the channel list
<i>channelList</i>	A string listing the channels and channel patterns to query to see which are forbidden to close

Details

The *channelList* parameter indicates which channels to check, and may include:

- `allslots` or `slotX` (where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B)
- Channel ranges or individual channels
- 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, row, column.

Example

```
channel.reset("slot1")
channel.setforbidden("1A01:1A05")
print(channel.getforbidden("allslots"))
print(channel.getforbidden("slot1"))
print(channel.getforbidden(
    "1A01:1A03,1B04,1B08,1B12"))
```

Reset the channels.

Set channels 1A01, 1A02, 1A03, 1A04, and 1A05 to be forbidden.

List the forbidden channels on all slots, slot 1, and list of channels.

Output:

```
1A01,1A02,1A03,1A04,1A05
```

```
1A01,1A02,1A03,1A04,1A05
```

```
1A01,1A02,1A03
```

Also see

[channel.clearforbidden\(\)](#) (on page 7-18)

[channel.setforbidden\(\)](#) (on page 7-45)

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	System reset Channel reset	Create configuration script	slot, row, column identifier

Usage

```
label = channel.getlabel(channelList)
```

<i>label</i>	A string listing the comma-delimited labels for items in <i>channelList</i>
<i>channelList</i>	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 for Model 707B, or 1 for Model 708B) or `allslots`.

- An error is generated if:
- The slot is empty
- The specified channel is not on the installed card
- A channel pattern is specified
- The channel does not support a label setting

Command processing stops as soon as an error is detected, and then a `nil` response is generated. No partial list of labels is returned.

Example

```
channel.reset("1A01")
print(channel.getlabel("1A01"))
channel.setlabel("1A01", "Device")
print(channel.getlabel("1A01"))
```

This example resets the channel, prints the default label of the channel, sets the label to "Device", and returns the new label.

Output:
1A01
Device

Also see

[channel.setlabel\(\)](#) (on page 7-46)

Data retrieval commands

channel.getlabelcolumn()

This function retrieves the label that was assigned to a column.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Empty string or blank

Usage

```
label = channel.getlabelcolumn(channelList)
```

<i>label</i>	A string that lists the comma-delimited labels for the items in <i>channelList</i>
<i>channelList</i>	A string that lists the channels to query for the labels associated with them

Details

The parameter *channelList* can contain more than one channel. Use a comma to delimit the labels for the channels. The return string *label* lists the labels in the same order that the channels are specified.

You cannot specify a channel pattern.

The *channelList* parameter can contain *slotX* (where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B) or *allslots*. It can also contain a label. However, if the label exists, it is in the returned response and not the numeric channel number.

Example

```
channel.setlabelcolumn("1A01", "DUT1")
channel.setlabelcolumn("1A03", "DUT2")
print(channel.getlabelcolumn("1a01:1a12"))
```

Label all the column labels on a card

Output:
DUT1,,DUT2,,,,,,,,,
Also note the change on the display

Also see

[channel.getlabelrow\(\)](#) (on page 7-33)

[channel.setlabelcolumn\(\)](#) (on page 7-47)

[channel.setlabelrow\(\)](#) (on page 7-49)

channel.getlabelrow()

This function retrieves the label assigned to a row.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Empty string or blank

Usage

```
label = channel.getlabelrow(channelList)
```

<i>label</i>	A string that lists the comma-delimited labels for the items in <i>channelList</i>
<i>channelList</i>	A string that lists the channels to query for the labels associated with them

Details

The parameter *channelList* can contain more than one channel. If it does, use a comma to delimit the labels for the channel rows. The return string *label* lists the labels in the same order that the channels are specified.

The *channelList* parameter can contain *slotX* (where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B) or *allslots*. It can also contain a label, but it cannot contain a channel pattern.

Example

```
channel.setrowlabel("1A01", "DUT1")
channel.setrowlabel("1C01", "DUT2")
print(channel.getlabelrow("1A01, 1B01,
    1C01, 1D01, 1E01, 1F01, 1G01, 1H01"))
```

Label the row labels on a card
Output:
DUT1,,DUT2,,,,,
Also note the change on the display

Also see

[channel.getlabelcolumn\(\)](#) (on page 7-32)

[channel.setlabelcolumn\(\)](#) (on page 7-47)

[channel.setlabelrow\(\)](#) (on page 7-49)

channel.getstate()

Queries the state indicators of the channels in the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Not saved	0

Usage

```
state = channel.getstate(channelList)
state = channel.getstate(channelList, indicatorMask)
```

<i>state</i>	Return string listing the comma-delimited states for the channels in <i>channelList</i>
<i>channelList</i>	String specifying the channels to query, using normal channel list syntax
<i>indicatorMask</i>	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). The optional state *indicatorMask* can be used to return only certain indicators. If there is no *indicatorMask*, then all indicators are returned.

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 has occurred when the `channel.getstate()` command was issued.

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.

The only state information is an indicator of relay state (`channel.IND_CLOSED`).

Example

```
print(channel.getstate("4A01:4B08"))

channel.pattern.setimage("1A01,2B02,3C03",
    "Path")
print(channel.getstate("Path"))
print(channel.getstate("3C03"))

-- Unmasking the return value must be done
-- one channel at a time.
if bit.band(channel.IND_CLOSED,
    tonumber(channel.getstate("4A10"))) == 1
then
    print("CLOSED")
else
    print("OPENED")
end
```

Queries the state of the first 20 channels on Slot 4.

See the state of channels in channel pattern called "PathList".

The `channel.IND_CLOSED` command equates to the number 1. Because the state is a bit-oriented value, you must perform a logical AND operation on the state to 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 7-27)

channel.gettype()

This function returns the type associated with a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not applicable	1

Usage

```
type = channel.gettype(channelList)
```

<i>type</i>	Returns a string listing the comma-delimited types for channels in <i>channelList</i>
<i>channelList</i>	String specifying the channels to query, using normal <i>channelList</i> syntax

Details

The channel type is defined by the physical hardware of the card on which the channel exists. The only valid channel type for the Models 707B and 708B is `channel.TYPE_SWITCH` or 1.

Example

```
print(channel.gettype("1A01"))
```

Queries the channel type of Row 1, Column 1, in Slot 1.

Also see

None

channel.open()

This function opens the specified channels and channel patterns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.open(channelList)
```

<i>channelList</i>	String listing the channels and channel patterns to open
--------------------	----------------------------------------------------------

Details

This function opens the specified channels for switching.

The settling time associated with a channel must elapse before the command completes. User delay is not added when a relay opens.

Example 1

```
channel.open("1A01:1A05, 3B03, Chans")
```

Opens channels in Row 1, Columns 01 to 05 on Slot 1; Row 2, Column 03 on Slot 3; and the channels in the channel pattern 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 7-19)
[channel.exclusiveclose\(\)](#) (on page 7-24)
[channel.close\(\)](#) (on page 7-19)
[channel.exclusiveclose\(\)](#) (on page 7-24)
[channel.exclusiveslotclose\(\)](#) (on page 7-25)
[channel.getclose\(\)](#) (on page 7-27)
[channel.getdelay\(\)](#) (on page 7-29)
[channel.pattern.getimage\(\)](#) (on page 7-38)
[channel.getstate\(\)](#) (on page 7-34)
[channel.setdelay\(\)](#) (on page 7-44)
[channel.setforbidden\(\)](#) (on page 7-45)

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
```

<i>name</i>	String representing the user-defined name of the channel pattern.
-------------	-------------------------------------------------------------------

Details

This function allows you to print or delete all user-created channel patterns in the runtime environment. The entries that are returned are listed in random order.

Example

```
channel.pattern.setimage("1A01,1A02",
    "patternA")
channel.pattern.setimage("1B01,1B02",
    "patternB")
channel.pattern.setimage("1C01,1C02",
    "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 = 1C01,1C02
patternA = 1A01,1A02
patternB = 1B01,1B02
```

Also see

- [channel.pattern.delete\(\)](#) (on page 7-38)
- [channel.pattern.getimage\(\)](#) (on page 7-38)
- [channel.pattern.setimage\(\)](#) (on page 7-39)
- [channel.pattern.snapshot\(\)](#) (on page 7-41)

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)
```

<i>name</i>	A string representing the name of the channel pattern to delete
-------------	-----------------------------------------------------------------

Details

An error is generated if the name does not exist.

Example

<code>channel.pattern.delete("Channels")</code>	Deletes a channel pattern called Channels
-------------------------------------------------	-------------------------------------------

Also see

- [channel.pattern.catalog\(\)](#) (on page 7-37)
- [channel.pattern.getimage\(\)](#) (on page 7-38)
- [channel.pattern.setimage\(\)](#) (on page 7-39)
- [channel.pattern.snapshot\(\)](#) (on page 7-41)

channel.pattern.getimage()

This function queries a channel pattern for associated channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Not applicable

Usage

```
channelList = channel.pattern.getimage(name)
```

<i>channelList</i>	A string specifying a list of channels that are represented by the name
<i>name</i>	A string representing the name of the channel pattern to query

Details

The returned string lists the channels in the slot, row, column format, even if a channel pattern was used to create it. Results for multiple channel patterns are delimited by a semicolon (;), and commas delimit the specific channels within a single channel pattern in the string.

Example

```
channel.pattern.setimage("1A01:1A05", "myPattern")
channel.pattern.setimage("1B01,1B03,1B05",
    "myRoute")

myImage = channel.pattern.getimage("myPattern")
print(myImage)
print(channel.pattern.getimage("myRoute"))
print(channel.pattern.getimage("myRoute,
    myPattern"))
```

Using a Model 7174 (or similar model) card in Slot 1, this example creates two channel patterns and then queries these patterns.

Output:

```
1A01,1A02,1A03,1A04,1A05
1B01,1B03,1B05
1B01,1B03,1B05;1A01,1A02,1
A03,1A04,1A05
```

Also see

- [channel.pattern.catalog\(\)](#) (on page 7-37)
- [channel.pattern.delete\(\)](#) (on page 7-38)
- [channel.pattern.setimage\(\)](#) (on page 7-39)

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	System reset Channel reset	Create configuration script	Not applicable

Usage

```
channel.pattern.setimage(channelList, name)
```

<i>channelList</i>	A string listing the channels and channel patterns to use when creating the new channel pattern
<i>name</i>	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 channel pattern is not created if an error is detected. You can create a channel pattern with an empty *channelList* parameter.

Once a channel pattern is created, the only way to add a channel to an existing pattern is to delete the old pattern and recreate the pattern with the new desired items. Alternatively, include the additional channels in the *channelList* with the channel pattern when using.



Channel patterns are lost when power is cycled.

Channel patterns are stored when you run the `createconfigscript()` command.

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 for Model 707B, or 1 for Model 708B) or `allslots`
- The name parameter contains a space character
- The pattern name exceeds 19 characters

Example

```
channel.pattern.setimage("1A01:1A10", "Channels")

oldList = channel.pattern.getimage("Channels")
newList = oldList .. ", 1C11"
channel.pattern.delete("Channels")
channel.pattern.setimage(newList, "Channels")
channel.close("Channels, 1D11")

Items = channel.pattern.getimage("Channels")
channel.pattern.setimage(Items, "Pattern")
channel.pattern.delete("Channels")
```

For this example, assume there is a Keithley Model 7174 or similar card in Slot 1.

Creates a pattern, appends a channel to the pattern by retrieving the pattern and recreating it, and then renames the pattern.

Also see

- [createconfigscript\(\)](#) (on page 7-50)
- [channel.pattern.catalog\(\)](#) (on page 7-37)
- [channel.pattern.delete\(\)](#) (on page 7-38)
- [channel.pattern.getimage\(\)](#) (on page 7-38)
- [channel.pattern.snapshot\(\)](#) (on page 7-41)

channel.pattern.snapshot()

This function creates a channel pattern.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Not applicable

Usage

```
channel.pattern.snapshot(name)
```

<i>name</i>	A string representing the name to associate with the present state of channels
-------------	--------------------------------------------------------------------------------

Details

This command stores the image of presently closed channels and associates them with the *name* parameter in memory.

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 previous image associated with the *name* is lost.

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.

Snapshots are lost when power is cycled. Channel patterns are stored when the `createsconfigript()` command is executed. Use the created script to restore them.

NOTE

Channel patterns are lost when power is cycled. Channel patterns are stored when the `createsconfigript()` command is executed. Use the created script to restore them. See `channel.pattern.setimage()` (on page 7-39) for valid *name* examples.

Example

```
channel.pattern.snapshot("voltagePath")
```

Creates a pattern named `voltagePath` that contains the presently closed channels.

Also see

[createconfigscript\(\)](#) (on page 7-50)

[channel.pattern.catalog\(\)](#) (on page 7-37)

[channel.pattern.delete\(\)](#) (on page 7-38)

[channel.pattern.getimage\(\)](#) (on page 7-38)

[channel.pattern.setimage\(\)](#) (on page 7-39)

channel.reset()

This function resets the specified channels to factory default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
channel.reset(channelList)
```

<i>channelList</i>	
	A string that lists the items to reset. The string can include: <ul style="list-style-type: none"> • allslots • slotX, where X is the slot number • channel patterns • channels, including a range of channels

Details

For the items specified in *channelList*, the following actions occur:

- Any closed channels open
- Additional user delay is set to zero (0)
- Labels are removed
- 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.

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

The rest of the 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("3A01:3A05")
```

Resets only row 1, columns 1 through 5 on Slot 3.

Example 4

```
channel.reset("5C05, 5D16")
```

Resets row 3, column 5, and row 4 column 16, on slot 5.

Also see

[channel functions and attributes](#) (on page 5-5)

[reset\(\)](#) (on page 7-140)

channel.setdelay()

This function sets additional delay time for specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	0

Usage

```
channel.setdelay(channelList, value)
```

<i>channelList</i>	A string listing the channels that need modifications to their delay time.
<i>value</i>	Desired delay time for items in <i>channelList</i> . Minimum is 0 seconds.

Details

The user delay is an additional delay that is added after a channel is closed. You can use this delay to allow additional settle time for a signal on that channel. For most cards, the resolution of the delay is 10 μ s. However, check the documentation for your card to verify. To see if the delay value was modified after setting, 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

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

<code>channel.setdelay("1A03, 1A05", 50e-6)</code>	Sets row 1 and columns 3 and 5 of slot 1 for a delay time of 50 microseconds.
----------------------------------------------------	-------------------------------------------------------------------------------

Example 2

<code>channel.setdelay("slot1", 0)</code>	Sets the channels on slot 1 for 0 delay time.
-------------------------------------------	-----------------------------------------------

Also see

[channel.getdelay\(\)](#) (on page 7-29)

channel.setforbidden()

This function prevents the closing of specified channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Not forbidden

Usage

```
channel.setforbidden(channelList)
```

<i>channelList</i>	A string that lists the channels to make forbidden to close.
--------------------	--------------------------------------------------------------

Details

The *channelList* parameter indicates the scope of channels affected and may include:

- allslots or slot*X* (where *X* equals 1 to 6 for Model 707B, or 1 for Model 708B)
- Channel ranges or individual channels

This function prevents all items contained in the channel list parameter from closing (applies the "forbidden to close" attribute to channels specified). To remove the "forbidden to close" attribute, use `channel.clearforbidden()`.

If a channel that is being set to forbidden is used in a channel pattern, the channel pattern is deleted when the channel is set to forbidden.

The channels in the *channelList* parameter must be installed in the instrument.

The *channelList* parameter cannot include a channel pattern.

If the scan list contains a channel that is forbidden, the scan list is invalidated.

Example

```
channel.pattern.setimage("1A01,1A02",
    "patternA")
channel.pattern.setimage("1B01,1B02",
    "patternB")
channel.pattern.setimage("1C01,1C02",
    "patternC")

for name in channel.pattern.catalog() do
    print(name .. " = " ..
        channel.pattern.getimage(name))
end

channel.setforbidden("1A02, 1B01")
for name in channel.pattern.catalog() do
    print(name .. " = " ..
        channel.pattern.getimage(name))
end
```

Create three channel patterns, and then print.

Set forbidden for one channel from patternA and patternB, then print the catalog again. Only patternC should remain.

Assuming no existing channel patterns, the output is:

```
patternC = 1C01,1C02
patternA = 1A01,1A02
patternB = 1B01,1B02
patternC = 1C01,1C02
```

Also see

[channel.clearforbidden\(\)](#) (on page 7-18)

[channel.getforbidden\(\)](#) (on page 7-30)

channel.setlabel()

This function sets the label associated with a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	No label

Usage

```
channel.setlabel(channelList, label)
```

<i>channelList</i>	A string that lists the channel to set the label associated with it
<i>label</i>	A string that represents the label for items in <i>channelList</i> , 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 *label* parameter must be unique. In addition, it cannot be the same as the name of a channel pattern, row label, or column label.

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("1A01", "start")
channel.close("start")
print(channel.getclose("allslots"))
```

Sets the label for channel row 1 and column 01 on slot 1 to "start".

Output:
1A01

Example 2

```
channel.setlabel("1A01", " ")
```

Clears the label for channel row 1 and column 01, slot 1 back to "1A01".

Example 3

```
channel.setlabel("1A01", " ")
```

Also clears the label for channel row 1 and column 01, slot 1 back to "1A01".

Also see

[channel functions and attributes](#) (on page 5-5)

[channel.setlabelcolumn\(\)](#) (on page 7-47)

[channel.setlabelrow\(\)](#) (on page 7-49)

channel.setlabelcolumn()

This function assigns a label to a column.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Empty string or blank

Usage

```
channel.setlabelcolumn(channel, label)
```

<i>channel</i>	A string that specifies the channel that contains the column to which the label will be applied
<i>label</i>	A string that lists the label for the channel column, up to eight characters

Details

A column label can be applied to columns of a matrix card. The label is then used on the display and can be accessed in a channel list (see [About channel commands](#) (on page 5-5)). For *channel*, you can specify any channel in the column. You cannot specify a channel pattern.

The row and column *label* parameters must be unique. In addition, they cannot be the same as the name of a channel pattern or channel label.

After defining a column label, you can use it with a row label to specify a channel instead of the channel specifier.

On the crosspoint display, the first four characters of the label are displayed. On the bottom display, the full label is displayed.

NOTE

Since a column label is common to all channels in that column, you only need to assign the column label to one channel.

Example 1

```
channel.setlabelrow("1B01", "SMU2")
channel.setlabelcolumn("1B02", "DUT2")
channel.close("SMU2+DUT2")
print(channel.getclose("allslots"))
```

Sets the label for the slot 1, row 2 to "SMU2" and slot 1, column 2 to "DUT2".
Use the labels to close a channel.
Output:
1B02

Example 2

```
channel.setlabelcolumn("1A01", "")
```

Clears the label for column 1 on slot 1 back to default.

Example 3

```
channel.setlabelcolumn("1A01", " ")
```

Also clears the label for column 1 on slot 1 back to default.

Example 4

```
channel.setlabelcolumn("2B01", "TwoC")
print(channel.getlabelcolumn("slot2"))
```

This example assumes a Model 7072 in slot 2.

Set the label to be TwoC, which assigns the label to all channels in the column. Output:

```
TwoC,,,,,,,,,,,,TwoC,,,,,,,,,,,,Tw
oC,,,,,,,,,,,,TwoC,,,,,,,,,,,,
TwoC,,,,,,,,,,,,TwoC,,,,,,,,,,,,
,TwoC,,,,,,,,,,,,TwoC,,,,,,,,
,,
```

Also see

[channel.getlabelcolumn\(\)](#) (on page 7-32)

[channel.getlabelrow\(\)](#) (on page 7-33)

[channel.setlabelrow\(\)](#) (on page 7-49)

channel.setlabelrow()

This function assigns a label to a row.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset	Create configuration script	Empty string or blank

Usage

```
channel.setlabelrow(channel, label)
```

<i>channel</i>	A string that specifies the channel that contains the row to which to apply a label
<i>label</i>	A string that lists the label for the channel row, up to eight characters

Details

A row label can be applied to columns of a matrix card. The label is then used on the display and can be accessed in a channel list (see [About channel commands](#) (on page 5-5)). For *channel*, you can specify any channel in the row. You cannot specify a channel pattern.

The row and column *label* parameters must be unique. In addition, they cannot be the same as the name of a channel pattern.

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

After defining a row label, you can use it to specify a channel instead of the default channel designation.

On the crosspoint display, the first four characters of the label are displayed. On the bottom display, the full label is displayed.

Labels can only be set for matrix cards.

NOTE

Since a row label is common to all channels in that row, you only need to assign the row label to one channel.

Example 1

```
channel.setlabelrow("1B01", "SMU2")
channel.setlabelcolumn("1B02", "DUT2")
channel.close("SMU2+DUT2")
print(channel.getclose("allslots"))
```

Sets the label for the slot 1, row 2 to "SMU2" and slot 1, column 2 to "DUT2".
Use the labels to close a channel.
Output:
1B02

Example 2

```
channel.setlabelrow("1A01", " ")
```

Clears the label for row 1 on slot 1 back to default.

Example 3

```
channel.setlabelrow("1A01", " ")
```

Also clears the label for row 1 on slot 1 back to default.

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
results = dataqueue.add(value)
results = dataqueue.add(value, timeout)
```

<i>results</i>	The resulting value of <code>true</code> or <code>false</code> based on the success of the function
<i>value</i>	The data item to add; <i>value</i> can be of any type
<i>timeout</i>	The maximum number of seconds to wait for space in the data queue

Details

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

<pre>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</pre>	<p>Clear the data queue.</p> <p>Each line adds one item to the data queue.</p> <p>Output: The dataqueue contains: 1.0000000e+01 1.1000000e+01 1.2000000e+01</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

- [dataqueue.CAPACITY](#) (on page 7-52)
- [dataqueue.clear\(\)](#) (on page 7-52)
- [dataqueue.count](#) (on page 7-53)
- [dataqueue.next\(\)](#) (on page 7-54)

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
```

<code>count</code>	The variable assigned the value of <code>dataqueue.CAPACITY</code>
--------------------	--------------------------------------------------------------------

Example

<pre>MaxCount = dataqueue.CAPACITY while dataqueue.count < MaxCount do dataqueue.add(1) end print("There are " .. dataqueue.count .. " items in the data queue")</pre>	<p>Add items to the data queue until it is at capacity.</p> <p>Output:</p> <p>There are 128 items in the data queue</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------

Also see

[dataqueue.add\(\)](#) (on page 7-51)
[dataqueue.clear\(\)](#) (on page 7-52)
[dataqueue.count](#) (on page 7-53)
[dataqueue.next\(\)](#) (on page 7-54)

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. The function 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 7-51)
- [dataqueue.CAPACITY](#) (on page 7-52)
- [dataqueue.count](#) (on page 7-53)
- [dataqueue.next\(\)](#) (on page 7-54)

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 off dataqueue.add() dataqueue.clear() dataqueue.next()	Not saved	Not applicable

Usage

```
count = dataqueue.count
```

<i>count</i>	The number of items in the data queue
--------------	---------------------------------------

Details

The count gets updated as entries are added and read from the data queue. It is also updated when the dataqueue is cleared.

The number of items in the data queue are controlled through `dataqueue.add()`, `dataqueue.next()`, and `dataqueue.clear()`, with a maximum of `dataqueue.CAPACITY` items.

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")
```

Add items to the data queue until it is at capacity.

Output:
 There are 128 items in the data queue
 There are 0 items in the data queue

Also see

[dataqueue.add\(\)](#) (on page 7-51)
[dataqueue.CAPACITY](#) (on page 7-52)
[dataqueue.clear\(\)](#) (on page 7-52)
[dataqueue.next\(\)](#) (on page 7-54)

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)
```

<i>value</i>	The next entry in the data queue
<i>timeout</i>	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

<pre>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")</pre>	<p>Clears the data queue, adds ten entries, then reads the entries from the data queue.</p> <p>Output:</p> <pre>There are 10 items in the data queue 1.000000000e+00 2.000000000e+00 3.000000000e+00 4.000000000e+00 5.000000000e+00 6.000000000e+00 7.000000000e+00 8.000000000e+00 9.000000000e+00 1.000000000e+01 There are 0 items in the data queue</pre>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[dataqueue.add\(\)](#) (on page 7-51)

[dataqueue.CAPACITY](#) (on page 7-52)

[dataqueue.clear\(\)](#) (on page 7-52)

[dataqueue.count](#) (on page 7-53)

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)`

<code>seconds</code>	The number of seconds to delay, maximum 100,000
----------------------	-------------------------------------------------

Details

It is impossible to delay for zero seconds.

The system delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the system to delay 5 μ s to 10 μ s (typical) more than the requested delay.

Example 1

<pre>beeper.beep(0.5, 2400) delay(0.250) beeper.beep(0.5, 2400)</pre>	Emit a double-beep at 2400 Hz. The sequence is 0.5 s on, 0.25 s off, 0.5 s on.
-----------------------------------------------------------------------	--------------------------------------------------------------------------------

Example 2

<pre>dataqueue.clear() dataqueue.add(35) timer.reset() delay(0.5) dt = timer.measure.t() print("Delay time was " .. dt) print(dataqueue.next())</pre>	<p>Clear the data queue, add 35 to it, then delay 0.5 seconds before reading it.</p> <p>Output: Delay time was 0.500099 3.500000000e+01</p>
-------------------------------------------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------

Also see

None

digio.readbit()

This function reads one digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readbit(N)
```

<i>data</i>	A custom variable that stores the state of the I/O line
<i>N</i>	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

<pre>print(digio.readbit(4))</pre>	Assume line 4 is set high, and it is then read. Output: 1.0000000e+00
------------------------------------	-----------------------------------------------------------------------------

Also see

- [Digital I/O port](#) (on page 2-7)
- [digio.readport\(\)](#) (on page 7-56)
- [digio.writebit\(\)](#) (on page 7-64)
- [digio.writeport\(\)](#) (on page 7-65)

digio.readport()

This function reads the digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = digio.readport()
```

<i>data</i>	The present value of the input lines on the digital I/O port
-------------	--------------------------------------------------------------

Details

The binary equivalent of the returned value indicates the value of the input lines on the I/O port. The least significant bit (0) of the binary number corresponds to Line 1. Bit 13 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

<pre>data = digio.readport() print(data)</pre>	Assume Lines 2, 4, 6, and 8 are set high when the I/O port is read. Output: 1.7000000e+02 This is binary 10101010
------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------

Also see

- [Digital I/O port](#) (on page 2-7)
- [digio.readbit\(\)](#) (on page 7-55)
- [digio.writebit\(\)](#) (on page 7-64)
- [digio.writeport\(\)](#) (on page 7-65)

digio.trigger[N].assert()

This function asserts a trigger on one of the digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

<i>N</i>	Digital I/O trigger line (1 to 14)
----------	------------------------------------

Details

The set pulsewidth determines how long the trigger is asserted.

Example

<code>digio.trigger[2].assert()</code>	Asserts a trigger on digital I/O line 2
----------------------------------------	-----------------------------------------

Also see

- [digio.trigger\[N\].pulsewidth](#) (on page 7-60)

digio.trigger[N].clear()

This function clears a trigger event on a digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].clear()
```

<i>N</i>	Digital I/O trigger line (1 to 14)
----------	------------------------------------

Details

The event detector of a trigger recalls if a trigger event has been detected since the last `digio.trigger[N].wait()` command. This function clears the event detector of a specified trigger line, discards the previous history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

```
digio.trigger[2].clear()
```

Clears the trigger event on I/O line 2.

Also see

[digio.trigger\[N\].overrun](#) (on page 7-60)

[digio.trigger\[N\].stimulus](#) (on page 7-62)

[digio.trigger\[N\].wait\(\)](#) (on page 7-64)

digio.trigger[N].EVENT_ID

This attribute identifies the trigger event generated by the digital I/O line *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = digio.trigger[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	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 describes the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Digital I/O trigger reset	Create configuration script	digio.TRIG_BYPASS

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

<i>triggerMode</i>	The trigger mode; see Details for values.
<i>N</i>	Digital I/O trigger line (1 to 14).

Details

Set *tmode* to one of the following values:

Trigger mode values	
<i>triggerMode</i>	Description
<code>digio.TRIG_BYPASS</code> or 0	Allows direct control of the line.
<code>digio.TRIG_FALLING</code> or 1	Detects falling-edge triggers as input; asserts a TTL-low pulse for output.
<code>digio.TRIG_RISING</code> or 2	If the programmed state of the line is high, the <code>digio.TRIG_RISING</code> mode behaves similar to <code>digio.TRIG_RISINGA</code> . If the programmed state of the line is low, the <code>digio.TRIG_RISING</code> mode behaves similar to <code>digio.TRIG_RISINGM</code> . This setting should only be used if necessary for compatibility with other Keithley Instruments products.
<code>digio.TRIG_EITHER</code> or 3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
<code>digio.TRIG_SYNCHRONOUSA</code> 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.
<code>digio.TRIG_SYNCHRONOUS</code> 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.
<code>digio.TRIG_SYNCHRONOUSA</code> or 6	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
<code>digio.TRIG_RISINGA</code> or 7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
<code>digio.TRIG_RISINGM</code> 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.

`digio.TRIG_SYNCHRONOUS` is provided for compatibility with older firmware. Either `digio.TRIG_SYNCHRONOUSA` or `digio.TRIG_SYNCHRONOUSA` should be used instead.

When reading the trigger mode, *tmode* is returned as a number.

To control the line state, use `digio.TRIG_BYPASS` with the `digio.writebit()` and the `digio.writeport()` commands.

Example

<code>digio.trigger[4].mode = 2</code>	Sets the trigger mode for I/O Line 4 to <code>digio.TRIG_RISING</code> .
----------------------------------------	--------------------------------------------------------------------------

Also see

[digio.trigger\[N\].reset\(\)](#) (on page 7-62)
[digio.writebit\(\)](#) (on page 7-64)
[digio.writeport\(\)](#) (on page 7-65)
[Scanning and triggering](#) (on page 3-1)

digio.trigger[N].overrun

Use this attribute to read the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	System reset Digital I/O trigger reset	Not saved	Not applicable

Usage

```
overrun = digio.trigger[N].overrun
```

<i>overrun</i>	Trigger overrun state (true or false)
<i>N</i>	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, this returns:
`false`

Also see

[digio.trigger\[N\].reset\(\)](#) (on page 7-62)

digio.trigger[N].pulsewidth

This attribute describes the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Digital I/O trigger reset	Create configuration script	10e-6

Usage

```
width = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = width
```

<i>width</i>	The pulse width (seconds)
<i>N</i>	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 μ s.

Also see

[digio.trigger\[N\].assert\(\)](#) (on page 7-57)
[digio.trigger\[N\].reset\(\)](#) (on page 7-62)
[digio.trigger\[N\].release\(\)](#) (on page 7-61)

digio.trigger[N].release()

This function releases an indefinite length or latched trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].release()
```

<i>N</i>	Digital I/O trigger line (1 to 14)
----------	------------------------------------

Details

Releases a trigger that was asserted with an indefinite pulse width time, as well as a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line (*N*) is affected.

Example

```
digio.trigger[4].release()
```

Releases digital I/O trigger Line 4.

Also see

[digio.trigger\[N\].pulsewidth](#) (on page 7-60)

digio.trigger[N].reset()

This function resets trigger values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].reset()
```

<i>N</i>	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`.

Also see

[digio.trigger\[N\].mode](#) (on page 7-58)

[digio.trigger\[N\].overrun](#) (on page 7-60)

[digio.trigger\[N\].pulsewidth](#) (on page 7-60)

[digio.trigger\[N\].stimulus](#) (on page 7-62)

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	System reset Digital I/O trigger reset	Create configuration script	0

Usage

```
triggerStimulus = digio.trigger[N].stimulus
digio.trigger[N].stimulus = triggerStimulus
```

<i>triggerStimulus</i>	The event identifier for the triggering event
<i>N</i>	Digital I/O trigger line (1 to 14)

Details

Set this attribute to zero (0) to disable the automatic trigger output.

Trigger stimulus for a digital I/O line may be set to one of the existing trigger event IDs, described in the following table.

Do not use the stimulus attribute for generating output triggers. Use `digio.trigger[N].assert()` instead.

Trigger event IDs**Trigger event ID**`digio.trigger[N].EVENT_ID`**Description**

An edge (either rising, falling, or either based on the configuration of the line) on the digital input line

`display.trigger.EVENT_ID`

The trigger key on the front panel is pressed

`trigger.EVENT_ID`

A *trg message on the active command interface. If GPIB is the active command interface, a GET message 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_SCAN_COMP`

Scan Complete Event

`scan.trigger.EVENT_IDLE`

Idle Event

NOTE

Use the name of the trigger event ID to set the stimulus value rather than the numeric value. Using the name makes the code compatible for future upgrades (for example, if the numeric values must change when enhancements are added to the instrument).

Example 1

```
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.

Example 2

```
digio.trigger[3].stimulus = 0
```

Clear the trigger stimulus of digital I/O Line 3.

Also see

[digio.trigger\[N\].assert\(\)](#) (on page 7-57)

[digio.trigger\[N\].clear\(\)](#) (on page 7-57)

[digio.trigger\[N\].reset\(\)](#) (on page 7-62)

digio.trigger[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = digio.trigger[N].wait(timeout)
```

<i>triggered</i>	Returns <code>true</code> if a trigger is detected, or <code>false</code> if no triggers are detected during the timeout period
<i>N</i>	Digital I/O trigger line (1 to 14)
<i>timeout</i>	Timeout in seconds

Details

This function pauses for up to *timeout* seconds for an input trigger. If one or more trigger events are detected since the last time `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 re-armed. 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 returns the output results.

Output if no trigger is detected:

```
false
```

Output if a trigger is detected:

```
true
```

Also see

[digio.trigger\[N\].clear\(\)](#) (on page 7-57)

digio.writebit()

This function sets a digital I/O line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writebit(N, data)
```

<i>N</i>	Digital I/O trigger line (1 to 14)
<i>data</i>	The value to write to the bit: <ul style="list-style-type: none"> 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

<code>digio.writebit(4, 0)</code>	Sets digital I/O Line 4 low (0).
-----------------------------------	----------------------------------

Also see

[digio.readbit\(\)](#) (on page 7-55)

[digio.readport\(\)](#) (on page 7-56)

[digio.trigger\[N\].mode](#) (on page 7-58)

[digio.writeport\(\)](#) (on page 7-65)

[digio.writeprotect](#) (on page 7-66)

digio.writeport()

This function writes to all digital I/O lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.writeport(data)
```

<i>data</i>	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 0000010101010. 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

<code>digio.writeport(255)</code>	Sets digital I/O Lines 1 through 8 high (binary 00000011111111).
-----------------------------------	------------------------------------------------------------------

Also see

[digio.readbit\(\)](#) (on page 7-55)
[digio.readport\(\)](#) (on page 7-56)
[digio.writebit\(\)](#) (on page 7-64)
[digio.writeprotect](#) (on page 7-66)

digio.writeprotect

This attribute describes the write-protect mask that protects bits from changes from the `digio.writebit()` and `digio.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset	Create configuration script	0

Usage

```
mask = digio.writeprotect
digio.writeprotect = mask
```

<i>mask</i>	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

<code>digio.writeprotect = 15</code>	Write-protects Lines 1, 2, 3, and 4
--------------------------------------	-------------------------------------

Also see

[digio.writebit\(\)](#) (on page 7-64)
[digio.writeport\(\)](#) (on page 7-65)

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, nonblocking commands. That is, the script does NOT wait for one of these commands to complete. These nonblocking 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 7-81)

[display.settext\(\)](#) (on page 7-82)

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()
```

<i>annunciators</i>	The bitmasked value showing 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 table below.

Annunciator (indicator) bitmasked values					
Indicator	Bit	Weighted value	Indicator	Bit	Weighted value
FILT	1	1	EDIT	9	256
MATH	2	2	ERR	10	512
4W	3	4	REM	11	1024
AUTO	4	8	TALK	12	2048
ARM	5	16	LSTN	13	4096
TRIG	6	32	SRQ	14	8192
*(star)	7	64	REAR	15	16384
SMPL	8	128	REL	16	32768

The following definitions exist:

`display.ANNUNCIATOR_X`

Where: *X* equals FILTER, MATH, 4_WIRE, AUTO, ARM, TRIG, STAR, SAMPLE, EDIT, ERROR, REMOTE, TALK, LISTEN, SRQ, REAR, or REL

The values correspond to the indicators listed above.

For example:

```
print(display.ANNUNCIATOR_EDIT)
2.560000000e+02
print(display.ANNUNCIATOR_TRIGGER)
3.200000000e+01
print(display.ANNUNCIATOR_AUTO)
8.000000000e+00
```

Example

<pre>myAnnunciators = display.getannunciators() print(myAnnunciators) rem = bit.bitand(myAnnunciators, 1024) if rem > 0 then print("REM is on") else print("REM is off") end</pre>	<p>Output:</p> <pre>1.2800000e+03 REM is on</pre> <p>REM indicator is turned on.</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------

Also see

[bit.bitand\(\)](#) (on page 7-9)

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()
```

<i>row</i>	<p>The row where the cursor is:</p> <ul style="list-style-type: none"> • 1: Top row • 2: Bottom row
<i>column</i>	<p>The column where the cursor is:</p> <ul style="list-style-type: none"> • If the cursor is in the top row: 1 to 20 • If the cursor is in the bottom row: 1 to 32
<i>style</i>	<p>Visibility of the cursor:</p> <ul style="list-style-type: none"> • 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

<pre>myRow, myColumn = display.getcursor() print(myRow, myColumn)</pre>	<p>This example reads the cursor position into local variables and prints them. An example return is:</p> <pre>1.000000000e+00 1.000000000e+00</pre>
-------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------

Example 2

<pre>print(display.getcursor())</pre>	<p>This example prints the cursor position directly. In this example, the cursor is in row 1 at column 3, with an invisible cursor:</p> <pre>1.000000000e+00 3.000000000e+00 0.000000000e+00</pre>
---------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

- [display.gettext\(\)](#) (on page 7-70)
- [display.screen](#) (on page 7-80)
- [display.setcursor\(\)](#) (on page 7-81)
- [display.setText\(\)](#) (on page 7-82)

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()
```

<code>keyCode</code>	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, `display.locallockout` must be used.

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, use 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.getlastkey()
print(key)
```

On the front panel, press the **MENU** key and then send the code to the left. This retrieves the key code for the last pressed key.

Output:
6.800000e+01

Also see

[display.locallockout](#) (on page 7-77)

[display.sendkey\(\)](#) (on page 7-80)

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)
```

<i>text</i>	The returned value, which contains the text that is presently displayed
<i>embellished</i>	Indicates whether to return: <ul style="list-style-type: none"> • Simple text: Set <i>embellished</i> to false • Text with embedded character codes: Set <i>embellished</i> to true
<i>row</i>	Selects the row from which to read the text. If <i>row</i> is not included, both rows of text are read The options are: <ul style="list-style-type: none"> • 1: Read from Row 1 • 2: Read from Row 2
<i>columnStart</i>	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
<i>columnEnd</i>	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

Sending 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 7-82).

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("ABCDEFGH IJ$DKLMNOPQRST")
display.setcursor(2, 1)
display.settext("abcdefghi jklm$Bnopqrstuvwxy z$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$Nabcdefghi jklmnopqrstuvwxy z123456
$RABCDEFGHIJKLMNOPQRST$N$Rabcdefghi jklm$Bnopqrstuvwxy z$F123456
abcdefghi jklmnopqrstuvwxy z123456
$Rijklm$Bnopqrstuvwxy z$F123456
ij
```

Example 2

```
display.clear()
display.settext("User Screen")
text = display.gettext()
print(text)
```

This returns 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 7-67)

[display.getcursor\(\)](#) (on page 7-68)

[display.setcursor\(\)](#) (on page 7-81)

[display.settext\(\)](#) (on page 7-82)

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)
```

<i>format</i>	A string that defines how the input field is formatted. See Details for more information.
<i>default</i>	The default value for the input value.
<i>minimum</i>	The minimum input value.
<i>maximum</i>	The maximum input value.

Details

format uses 0s, the decimal point, polarity sign, and exponents to define how the input field is formatted. *format* can include the options shown in the following table.

format options

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 0s.	+00.0000e+00
.	Include to have a decimal point appear in the value.	+0.00

default is the value shown when the value is first displayed.

minimum and *maximum* can be used to limit the values that can be entered. When + is selected for *format*, the minimum limit must be more than or equal to zero. When limits are used, the operator cannot enter values above or below these limits.

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 input value is limited to $\pm 1e37$.

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 the operator enters a value and presses the **ENTER** key.

For positive and negative entry ("+" sign used for the value field and/or the exponent field), polarity of a non-zero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After sending this command and pressing the **EXIT (LOCAL)** key, value 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 = 1.350000000e+00
```

Also see

[display.prompt\(\)](#) (on page 7-78)

[display.setcursor\(\)](#) (on page 7-81)

[display.settext\(\)](#) (on page 7-82)

display.loadmenu.add()

This function adds an entry to the User 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.add(displayName, code)
display.loadmenu.add(displayName, code, memory)
```

<i>displayName</i>	The name that is added to the User menu.
<i>code</i>	The code that is run from the User menu.
<i>memory</i>	Determines if code is saved to nonvolatile memory: 0 or <code>display.DONT_SAVE</code> : Does not save the code to nonvolatile memory. 1 or <code>display.SAVE</code> : 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** to select from the available code to load.

After loading the code, you can execute it when the **RUN** key is pressed.

You can add items in any order. They are always displayed in alphabetic order when the menu is selected.

The code can be made up of scripts, functions, variables, and commands. If *memory* is set to `display.SAVE`, commands are saved with the code in nonvolatile memory. Scripts, functions, and variables used in the code are not saved by `display.SAVE`. Functions and variables need to be saved along with the script. If the script is not saved in nonvolatile memory, it will be lost when the Models 707B and 708B are 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 runtime 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 7-76)

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 ... end
for displayName, code in display.loadmenu.catalog() do ... end
```

<i>displayName</i>	The name displayed in the LOAD menu
<i>code</i>	The code associated with the <i>displayName</i>

Details

Each time through the loop, *displayName* and *code* will take on the values in the User menu. The list that is returned is in random order.

Example

<pre>for myNames, listCode in display.loadmenu.catalog() do print(myNames, listCode) end</pre>	Output: Test DUT1() beeper.beep(2, 500) Part1 testpart([[Part1]], 5.0) Test9 Test9()
----------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------

Also see

[display.loadmenu.add\(\)](#) (on page 7-74)
[display.loadmenu.delete\(\)](#) (on page 7-76)

display.loadmenu.delete()

This function removes an entry from the User 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)
```

<i>displayName</i>	The name to be deleted from the User menu
--------------------	-------------------------------------------

Details

If you delete an entry from the User menu, you can no longer run it by pressing the **LOAD** key.

Example

<pre>display.loadmenu.delete("Test9") for displayName, code in display.loadmenu.catalog() do print(displayName, code) end</pre>	Deletes the script named "Test9" Output: Test DUT1() beeper.beep(2, 500) Part1 testpart([[Part1]], 5.0)
-------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------

Also see

[display.loadmenu.add\(\)](#) (on page 7-74)

[display.loadmenu.catalog\(\)](#) (on page 7-76)

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	display.UNLOCK

Usage

```
lockout = display.locallockout
display.locallockout = lockout
```

<i>lockout</i>	display.UNLOCK or 0: Unlocks EXIT key display.LOCK or 1: Locks out EXIT 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/remote operation.

Example

<code>display.locallockout = display.LOCK</code>	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)
```

<i>selection</i>	Name of the variable that holds the menu item selected
<i>name</i>	Menu name to display on the top line
<i>items</i>	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 **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)
```

<i>format</i>	A string that defines how the input field is formatted. See Details for more information
<i>units</i>	Set the units text string for the top line (eight characters maximum). This is intended to indicate the units (for example, "V" or "A") for the value.
<i>help</i>	Text string to display on the bottom line (32 characters maximum).
<i>default</i>	The value that is shown when the value is first displayed.
<i>minimum</i>	The minimum input value that can be entered.
<i>maximum</i>	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
```

format uses 0s, the decimal point, polarity sign and exponents to define how the input field is formatted.

format can include the options shown in the following table.

format options

Option	Description	Examples
E	Include the E to display the value exponentially. Include a "+" sign for positive/negative exponent entry. Do not include the "+" 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 "+" 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 0s.	+00.0000E+00
.	The decimal point where needed for the value.	+0.00

minimum and *maximum* can be used to limit the values that can be entered. When + is selected for *format*, the minimum limit must be more than or equal to zero. When limits are used, the operator cannot enter values above or below these limits.

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 input value is limited to $\pm 1e37$.

After sending this command, script execution pauses for the operator to enter a value and press **ENTER**.

For positive and negative entry ("+" sign used for the value field and/or the exponent field), polarity of a non-zero value or exponent can be toggled by positioning the cursor on the polarity sign and turning the navigation wheel. Polarity will also toggle when using the navigation wheel to decrease or increase the value or exponent past zero. A zero value or exponent (for example, +00) is always positive and cannot be toggled to negative polarity.

After you send this command and press 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:

```
7.000000000e-01
```

Also see

[display.inputvalue\(\)](#) (on page 7-73)

display.screen

This attribute describes the selected display screen.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset	Create configuration script	display.MAIN

Usage

```
displayID = display.screen
display.screen = displayID
```

<i>displayID</i>	One of the following values: <ul style="list-style-type: none"> 1 or <code>display.MAIN</code>: Displays the main screen 2 or <code>display.USER</code>: 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

<code>display.screen = display.USER</code>	Selects the user display.
--------------------------------------------	---------------------------

Also see

[display.settext\(\)](#) (on page 7-82)

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)
```

<i>keyCode</i>	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.

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

```
display.sendkey(display.KEY_RUN)    Simulates pressing the RUN key.
```

Also see

[Front-panel keys](#) (on page 2-18)

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)
```

<i>row</i>	The row number for the cursor (1 or 2).
<i>column</i>	The active column position to set; row 1 has columns 1 to 20, row 2 has columns 1 to 32.
<i>style</i>	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, nonblocking commands. That is, the script does NOT wait for one of these commands to complete. These nonblocking 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 the message "Hello World" on the instrument front panel, approximately center. Note that the top line of text is larger than the bottom.

Also see

[display.clear\(\)](#) (on page 7-67)

[display.getcursor\(\)](#) (on page 7-68)

[display.gettext\(\)](#) (on page 7-70)

[display.screen](#) (on page 7-80)

[display.settext\(\)](#) (on page 7-82)

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)
```

<i>text</i>	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, nonblocking commands. That is, the script does NOT wait for one of these commands to complete. These nonblocking 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 is left at the end of the line.

The text remains on the display until replaced or cleared.

The following character codes can be also be included in the text string:

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, non-blinking
\$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 7-67)

[display.getcursor\(\)](#) (on page 7-68)

[display.gettext\(\)](#) (on page 7-70)

[display.screen](#) (on page 7-80)

[display.setcursor\(\)](#) (on page 7-81)

display.trigger.EVENT_ID

This constant describes the trigger event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = display.trigger.EVENT_ID
```

<i>eventID</i>	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 events.

Also see

None

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()
```

<i>keyCode</i>	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 *keyCode* value for that 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 7-69)
- [display.sendkey\(\)](#) (on page 7-80)
- [display.settext\(\)](#) (on page 7-82)

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()
```

Also see

- [errorqueue.count](#) (on page 7-86)
- [errorqueue.next\(\)](#) (on page 7-86)
- [Status model](#) (on page C-1)

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
```

<code>count</code>	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:
4.0000000e+00

Also see

[errorqueue.clear\(\)](#) (on page 7-85)

[errorqueue.next\(\)](#) (on page 7-86)

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()
```

<code>errorCode</code>	The error code number for the entry.
<code>message</code>	The message that describes the error code.
<code>severity</code>	The severity level (0, 10, 20, 30, or 40). See Details for more information.
<code>errorNode</code>	The node number where the error originated.

Details

Entries are stored in a first-in, first-out (FIFO) queue.

Error codes and messages are listed in order of occurrence.

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* returns the node number where the error originated.

Example

<pre>errorcode, message = errorqueue.next() print(errorcode, message)</pre>	<p>Reads the oldest entry in the error queue. The output below indicates that the queue is empty.</p> <p>Output: 0.0000000e+00 Queue Is Empty</p>
-----------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

- [errorqueue.clear\(\)](#) (on page 7-85)
- [errorqueue.count](#) (on page 7-86)
- [Error and status messages](#) (on page 8-1)
- [Status model](#) (on page C-1)

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()
```

<i>logString</i>	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())
```

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 7-88)
- [eventlog.count](#) (on page 7-89)
- [eventlog.enable](#) (on page 7-90)
- [eventlog.next\(\)](#) (on page 7-90)
- [eventlog.overwritemethod](#) (on page 7-91)

eventlog.clear()

This function removes all entries from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Also see

- [eventlog.all\(\)](#) (on page 7-87)
- [eventlog.count](#) (on page 7-89)
- [eventlog.enable](#) (on page 7-90)
- [eventlog.next\(\)](#) (on page 7-90)
- [eventlog.overwritemethod](#) (on page 7-91)

eventlog.count

This attribute gets the number of events contained in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	System 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)
```

Display the present number of events contained the Model 707B or 708B event log.

Output looks similar to:
3.0000000e000

Also see

- [eventlog.all\(\)](#) (on page 7-87)
- [eventlog.clear\(\)](#) (on page 7-88)
- [eventlog.enable](#) (on page 7-90)
- [eventlog.next\(\)](#) (on page 7-90)
- [eventlog.overwritemethod](#) (on page 7-91)

eventlog.enable

This attribute enables or disables the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset	Create configuration script	eventlog.ENABLE

Usage

```
status = eventlog.enable
eventlog.enable = status
```

<i>status</i>	<p>The enable status of the event log:</p> <ul style="list-style-type: none"> eventlog.ENABLE or 1: Event log enable eventlog.DISABLE or 0: 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

<pre>print(eventlog.enable) eventlog.enable = eventlog.DISABLE print(eventlog.enable)</pre>	<p>Displays the present status of the Model 707B or 708B event log.</p> <p>Output:</p> <pre>1.0000000e00 0.0000000e00</pre>
---------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------

Also see

- [eventlog.all\(\)](#) (on page 7-87)
- [eventlog.clear\(\)](#) (on page 7-88)
- [eventlog.count](#) (on page 7-89)
- [eventlog.next\(\)](#) (on page 7-90)
- [eventlog.overwritemethod](#) (on page 7-91)

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()
```

<i>logString</i>	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())
```

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())
```

Output:

```
17:28:25.549 10 Oct 2009, LAN6, 192.168.1.102, LXI, 0, 1192037135, <no time>, 0,
0x0
```

Example 3

```
print(eventlog.next())
```

Output:

```
17:28:31.563 10 Oct 2009, LAN4, 192.168.1.102, LXI, 0, 1192037136, <no time>, 0,
0x0
```

Example 4

```
print(eventlog.next())
```

Output:

```
nil
```

Also see

- [eventlog.all\(\)](#) (on page 7-87)
- [eventlog.clear\(\)](#) (on page 7-88)
- [eventlog.count](#) (on page 7-89)
- [eventlog.enable](#) (on page 7-90)
- [eventlog.overwritemethod](#) (on page 7-91)

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	System reset	Create configuration script	eventlog.DISCARD_OLDEST

Usage

```
method = eventlog.overwritemethod
eventlog.overwritemethod = method
```

<i>method</i>	Set to one of the following values: <ul style="list-style-type: none"> • 0 or <code>eventlog.DISCARD_NEWEST</code>: New entries are not logged • 1 or <code>eventlog.DISCARD_OLDEST</code>: Old entries are deleted as new events are logged
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

When this attribute is set to `eventlog.DISCARD_NEWEST`, new entries are not be logged.

When this attribute is set to `eventlog.DISCARD_OLDEST`, the oldest entry is discarded when a new entry is added.

Example

<code>eventlog.overwritemethod = 0</code>	When the log is full, the event log will ignore new entries.
-------------------------------------------	--------------------------------------------------------------

Also see

[eventlog.all\(\)](#) (on page 7-87)
[eventlog.clear\(\)](#) (on page 7-88)
[eventlog.count](#) (on page 7-89)
[eventlog.enable](#) (on page 7-90)
[eventlog.next\(\)](#) (on page 7-90)

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 7-243)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers printed with the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	System reset	Create configuration script	10

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

<i>precision</i>	A number representing the number of digits to be printed for numbers with the <code>print()</code> , <code>printbuffer()</code> , and <code>printnumber()</code> 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.ascii.precision` attribute is only used with the ASCII format.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x)</pre>	Output: <pre>2.540000000e+00 2.54e+00</pre>
--------------------------------------------------------------------------------------------------------	-------------------------------------------------------

Also see

[format.byteorder](#) (on page 7-93)
[format.data](#) (on page 7-94)
[print\(\)](#) (on page 7-138)
[printbuffer\(\)](#) (on page 7-138)
[printnumber\(\)](#) (on page 7-140)

format.byteorder

This attribute sets the binary byte order for data printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	System reset	Create configuration script	format.LITTLEENDIAN

Usage

```
order = format.byteorder
format.byteorder = order
```

<code>order</code>	Byte order value as follows: <ul style="list-style-type: none"> • Most significant byte first: 0, <code>format.NORMAL</code>, <code>format.NETWORK</code>, or <code>format.BIGENDIAN</code> • Least significant byte first: 1, <code>format.SWAPPED</code> or <code>format.LITTLEENDIAN</code>
--------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

This attribute selects the byte order in which data is written when 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.

NOTE

Binary formats are not intended to be human readable.

Example

<pre>x = 1.23 format.data = format.REAL32 format.byteorder = format.LITTLEENDIAN printnumber(x) format.byteorder = format.BIGENDIAN printnumber(x)</pre>	Output depends on the terminal program you use, but will look something like: <pre>#0qp?? #0??p□</pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------------------------------

Also see

[format.asciiprecision](#) (on page 7-93)

[format.data](#) (on page 7-94)

[printbuffer\(\)](#) (on page 7-138)

[printnumber\(\)](#) (on page 7-140)

format.data

This attribute sets the data format for data printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	System reset	Create configuration script	format.ASCII

Usage

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

<i>value</i>	<p>The format to use for data, set to one of the following values:</p> <ul style="list-style-type: none"> • ASCII format: 1 or <code>format.ASCII</code> • Single-precision IEEE Std 754 binary format: 2, <code>format.SREAL</code>, or <code>format.REAL32</code> • Double-precision IEEE-754 binary format: 3, <code>format.REAL</code>, <code>format.REAL64</code>, or <code>format.DREAL</code>
--------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

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.

The IEEE Std 754 binary formats use four bytes each for single-precision values and eight bytes each 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 human readable.

Example

<pre>format.asciiprecision = 10 x = 3.14159265 format.data = format.ASCII printnumber(x) format.data = format.REAL64 printnumber(x)</pre>	<pre>3.141592650e+00 #0ñÔËŠú! @</pre>
-------------------------------------------------------------------------------------------------------------------------------------------	--------------------------------------------

Also see

[format.asciiprecision](#) (on page 7-93)

[format.byteorder](#) (on page 7-93)

[printbuffer\(\)](#) (on page 7-138)

[printnumber\(\)](#) (on page 7-140)

gettimezone()

This function retrieves the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timeZone = gettimezone()
```

<i>timeZone</i>	The local timezone of the instrument
-----------------	--------------------------------------

Details

See `settimezone()` for additional details on the time zone format and a description of the fields. *timeZone* can be in either of the following formats:

- If one argument was used with `settimezone()`, the format used is:
GMThh:mm:ss
- If four arguments 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

<code>timezone = gettimezone()</code>	Reads the value of the local timezone.
---------------------------------------	----------------------------------------

Also see

[settimezone\(\)](#) (on page 7-168)

gpib.address

This attribute describes the GPIB address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	16

Usage

```
address = gpib.address
gpib.address = address
```

<i>address</i>	GPIB address (0 to 30)
----------------	------------------------

Details

The change to the GPIB address takes effect when the command is processed. If there are response messages in the output queue when this command is processed, they must be read at the new address.

Any response messages generated after processing this command are sent with the new settings. If command messages are being queued (sent before this command has executed), the new settings may take effect in the middle of a subsequent command message, so care should be exercised when setting this attribute from the 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 GPIB address is stored in nonvolatile memory. The reset function has no effect on the address.

Example

<pre>gpib.address = 26 address = gpib.address print(address)</pre>	<p>Sets the GPIB address and reads the address. Output: 2.60000000e+01</p>
--------------------------------------------------------------------	------------------------------------------------------------------------------------

Also see

[GPIB](#) (on page 2-31)

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 current configuration settings.

This function initiates a background operation. LAN configuration could be a lengthy operation. Although the function returns immediately, the LAN initialization will continue 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 DHCP or 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
```

<i>dnsAddress</i>	DNS server IP address
<i>N</i>	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.

Unused entries will be returned as "0.0.0.0" when read. *dnsAddress* must be a string specifying the DNS server's IP address in dotted decimal notation. To disable an entry, set its value to "0.0.0.0" or the empty string "".

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. The IP address obtained from the DHCP server takes priority for all DNS lookups.

Example

```
dnsaddress = "164.109.48.173"
lan.config.dns.address[1] = dnsaddress
```

Write a DNS address of 164.109.48.173 as address 1.
Set the DNS address to *dnsaddress*

Also see

- [lan.config.dns.domain](#) (on page 7-99)
- [lan.config.dns.dynamic](#) (on page 7-100)
- [lan.config.dns.hostname](#) (on page 7-101)
- [lan.config.dns.verify](#) (on page 7-101)
- [lan.restoredefaults\(\)](#) (on page 7-107)

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
```

<i>domain</i>	Dynamic DNS registration domain; use a string of 255 characters or fewer
---------------	--------------------------------------------------------------------------

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)
```

Returns 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 7-100)

[lan.config.dns.hostname](#) (on page 7-101)

[lan.config.dns.verify](#) (on page 7-101)

[lan.restoredefaults\(\)](#) (on page 7-107)

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	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:

- lan.ENABLE or 1: Enabled
- lan.DISABLE or 0: 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)
```

Returns the dynamic registration state.

If dynamic DNS registration is enabled, the response is:

```
1.0000000e+00
```

Also see

[lan.config.dns.hostname](#) (on page 7-101)

[lan.restoredefaults\(\)](#) (on page 7-107)

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	LAN restore defaults	Nonvolatile memory	Instrument specific (see Details)

Usage

```
hostName = lan.config.dns.hostname
lan.config.dns.hostname = hostName
```

<i>hostName</i>	The host name to use for dynamic DNS registration. The host name must: <ul style="list-style-type: none"> • be a string of 255 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 format 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-707B-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 255 characters can be entered here, care must be taken to be sure the combined length is also no more than 255 characters.

Example

```
print(lan.config.dns.hostname) Returns the present dynamic DNS host name.
```

Also see

[lan.config.dns.dynamic](#) (on page 7-100)

[lan.restoredefaults\(\)](#) (on page 7-107)

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	lan.ENABLE

Usage

```
state = lan.config.dns.verify
lan.config.dns.verify = state
```

<i>state</i>	DNS hostname verification state: <ul style="list-style-type: none"> lan.ENABLE or 1: DNS host name verification enabled lan.DISABLE or 0: 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

<code>print(lan.config.dns.verify)</code>	Returns the present DNS host name verification state. If it is enabled, the return is: 1.0000000e+00
-------------------------------------------	----------------------------------------------------------------------------------------------------------------

Also see

[lan.config.dns.hostname](#) (on page 7-101)

[lan.restoredefaults\(\)](#) (on page 7-107)

lan.config.gateway

This attribute describes 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
```

<i>gatewayAddress</i>	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 currently in effect. Use the `lan.status` attributes to determine the current operating state of the LAN.

The IP address must be formatted in four groups of numbers each separated by a decimal.

Example

<code>print(lan.config.gateway)</code>	Returns the default gateway address. For example, you might see the output: 010.060.008.001
----------------------------------------	------------------------------------------------------------------------------------------------

Also see

[lan.status.gateway](#) (on page 7-110)

[lan.restoredefaults\(\)](#) (on page 7-107)

lan.config.ipaddress

This attribute describes the LAN IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	"0.0.0.0"

Usage

```
ipAddress = lan.config.ipaddress
lan.config.ipaddress = ipAddress
```

<i>ipAddress</i>	LAN IP address. Must be a string specifying the IP address in dotted decimal notation.
------------------	----------------------------------------------------------------------------------------

Details

This attribute 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 currently in effect. Use the `lan.status` attributes to determine the current operating state of the LAN.

Example

```
ipaddress = lan.config.ipaddress
```

Returns the presently set LAN IP address.

Also see

[lan.restoredefaults\(\)](#) (on page 7-107)

[lan.status.ipaddress](#) (on page 7-110)

lan.config.method

This attribute describes the LAN settings configuration method.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	lan.AUTO

Usage

```
method = lan.config.method
lan.config.method = method
```

<i>method</i>	<p>The method for configuring LAN settings. It can be one of the following values:</p> <ul style="list-style-type: none"> <code>lan.AUTO</code> or <code>1</code>: Selects automatic sequencing of configuration methods <code>lan.MANUAL</code> or <code>0</code>: 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

<pre>print(lan.config.method)</pre>	Returns the current method.
	For example: 1.0000000e+00

Also see

[lan.restoredefaults\(\)](#) (on page 7-107)

lan.config.subnetmask

This attribute describes 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
```

<i>mask</i>	LAN subnet mask value string specifying the subnet mask in dotted decimal notation
-------------	------------------------------------------------------------------------------------

Details

This attribute specifies the LAN subnet mask to use when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting currently in effect. Use the `lan.status.subnetmask` attribute to determine the current operating state of the LAN.

Example

<pre>print(lan.config.subnetmask)</pre>	Returns the LAN subnet mask, such as: 255.255.255.000
-----------------------------------------	----------------------------------------------------------

Also see

[lan.restoredefaults\(\)](#) (on page 7-107)

[lan.status.subnetmask](#) (on page 7-114)

lan.lxidomain

This attribute describes 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
```

<code>domain</code>	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

<code>print(lan.lxidomain)</code>	Displays the LXI domain.
-----------------------------------	--------------------------

Also see

[lan.restoredefaults\(\)](#) (on page 7-107)

lan.nagle

This attribute describes the use of the LAN Nagle algorithm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	LAN restore defaults	Nonvolatile memory	lan.ENABLE

Usage

```
state = lan.nagle
lan.nagle = state
```

<code>state</code>	<ul style="list-style-type: none"> lan.ENABLE: Enable the LAN Nagle algorithm for TCP connections 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 7-107)

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 7-98)

[lan.restoredefaults\(\)](#) (on page 7-107)

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 on default	
ICL attribute	Default setting
<code>lan.config.dns.address[N]</code>	"0.0.0.0"
<code>lan.config.dns.domain</code>	" "
<code>lan.config.dns.dynamic</code>	<code>lan.ENABLE</code>
<code>lan.config.dns.hostname</code>	"K-<model number>-<serial number>"
<code>lan.config.dns.verify</code>	<code>lan.ENABLE</code>
<code>lan.config.gateway</code>	"0.0.0.0"
<code>lan.config.ipaddress</code>	"0.0.0.0"
<code>lan.config.method</code>	<code>lan.AUTO</code>
<code>lan.config.subnetmask</code>	"255.255.255.0"
<code>lan.lxidomain</code>	0
<code>localnode.password</code>	"admin"

This command is run when `lan.reset()` is sent.

Example

<code>lan.restoredefaults()</code>	Restores the LAN defaults.
------------------------------------	----------------------------

Also see

[lan.reset\(\)](#) (on page 7-107)

[localnode.password](#) (on page 7-129)

lan.status.dns.address[N]

Reads 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]
```

<i>dnsAddress</i>	DNS server IP address
<i>N</i>	Entry index (1, 2, or 3)

Details

This attribute is an array of DNS server addresses. The system 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 here, 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

<code>print(lan.status.dns.address[1])</code>	Returns DNS server address 1, such as: 164.109.48.173
-----------------------------------------------	----------------------------------------------------------

Also see

[lan.status.dns.name](#) (on page 7-109)

lan.status.dns.name

Reads 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
```

<i>hostName</i>	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

<code>print(lan.status.dns.name)</code>	Returns the dynamic DNS host name.
-----------------------------------------	------------------------------------

Also see

[lan.config.dns.address\[N\]](#) (on page 7-98)

[lan.config.dns.hostname](#) (on page 7-101)

lan.status.duplex

This attribute describes which duplex mode is currently 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
```

<i>duplex</i>	LAN duplex setting can be one of the following values: <ul style="list-style-type: none"> lan.FULL or 1: full-duplex operation lan.HALF or 0: half-duplex operation
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example

<code>print(lan.status.duplex)</code>	Returns the present LAN duplex mode, such as: 1.0000000e+00
---------------------------------------	----------------------------------------------------------------

Also see

None

lan.status.gateway

This attribute reads the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
gatewayAddress = lan.status.gateway
```

<code>gatewayAddress</code>	LAN default gateway address
-----------------------------	-----------------------------

Details

The value of `gatewayAddress` is a string that indicates the IP address of the default gateway in dotted decimal notation.

Example

<pre>print(lan.status.gateway)</pre>	Returns the gateway address, such as: 10.60.8.1
--------------------------------------	----------------------------------------------------

Also see

[lan.config.gateway](#) (on page 7-102)

lan.status.ipaddress

This attribute reads the LAN IP address currently in use.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
ipAddress = lan.status.ipaddress
```

<code>ipAddress</code>	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

<pre>print(lan.status.ipaddress)</pre>	Returns the LAN IP address currently in use, such as: 10.60.8.83
----------------------------------------	---------------------------------------------------------------------

Also see

[lan.config.ipaddress](#) (on page 7-104)

lan.status.macaddress

This attribute returns 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
```

<i>macAddress</i>	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

<pre>print(lan.status.macaddress)</pre>	Returns the MAC address of the instrument, for example: 00:60:1A:00:00:57
-----------------------------------------	------------------------------------------------------------------------------

Also see

None

lan.status.port.dst

This attribute reads 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
```

<i>port</i>	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

<pre>print(lan.status.port.dst)</pre>	Returns the LAN dead socket termination port number, such as: 5.0300000e+03
---------------------------------------	--------------------------------------------------------------------------------

Also see

None

lan.status.port.rawsocket

This attribute reads 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
```

<code>port</code>	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

<code>print(lan.status.port.rawsocket)</code>	Returns the Model 707B or 708B raw socket port number, such as: 5.025000000e+03
-----------------------------------------------	------------------------------------------------------------------------------------

Also see

None

lan.status.port.telnet

This attribute gets 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
```

<code>port</code>	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:
2.3000000e+01

Also see

None

lan.status.port.vxi11

This attribute gets 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
```

<i>port</i>	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)
```

Returns the VXI-11 number, such as:

1.024000000e+03

Also see

None

lan.status.speed

This attribute reads 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
```

<i>speed</i>	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)
```

Returns the instrument's transmission speed presently in use, such as:
1.0000000e+02

Also see

None

lan.status.subnetmask

This attribute reads the LAN subnet mask that is presently in use.

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.

Example

```
print(lan.status.subnetmask)
```

Returns the subnet mask of the instrument that is presently in use, such as:
255.255.255.0

Also see

None

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 trigger packet over LAN to assert (1 to 8)

Details

Generates and sends a LAN trigger packet for the LAN event number specified.

Sets the pseudostate to the appropriate state.

The following indexes provide the listed events:

- 1:LAN0
- 2:LAN1
- 3:LAN2
- ...
- 8:LAN7

Example

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

Creates a trigger with LAN packet 5.

Also see

[lan.trigger\[N\].clear\(\)](#) (on page 7-115)

[lan.trigger\[N\].mode](#) (on page 7-119)

[lan.trigger\[N\].overrun](#) (on page 7-121)

[lan.trigger\[N\].stimulus](#) (on page 7-123)

[lan.trigger\[N\].wait\(\)](#) (on page 7-125)

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 trigger packet over the LAN to clear (1 to 8)

Details

A trigger's event detector remembers if an event has been detected since the last call. This function clears a trigger's 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 7-114)
[lan.trigger\[N\].overrun](#) (on page 7-121)
[lan.trigger\[N\].stimulus](#) (on page 7-123)
[lan.trigger\[N\].wait\(\)](#) (on page 7-125)

lan.trigger[N].connect()

This function connects the `lan.trigger` instance to the listeners specified by protocol and IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.trigger[N].connect()
```

<i>N</i>	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 7-114)
[lan.trigger\[N\].ipaddress](#) (on page 7-119)
[lan.trigger\[N\].overrun](#) (on page 7-121)
[lan.trigger\[N\].protocol](#) (on page 7-122)
[lan.trigger\[N\].stimulus](#) (on page 7-123)
[lan.trigger\[N\].wait\(\)](#) (on page 7-125)

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
```

<i>connected</i>	The LAN event connection state: <ul style="list-style-type: none"> • true: Connected • false: Not connected
<i>N</i>	The LAN event number (1 to 8)

Details

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 set to false.

Set to 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)
```

Returns true if connected, or false if not connected.
Example output:
false

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 7-116)

[lan.trigger\[N\].ipaddress](#) (on page 7-119)

[lan.trigger\[N\].protocol](#) (on page 7-122)

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()
```

<i>N</i>	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 7-119)

[lan.trigger\[N\].protocol](#) (on page 7-122)

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
```

<i>N</i>	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

<pre>digio.trigger[14].stimulus = lan.trigger[1].EVENT_ID</pre>	Route an occurrence of a trigger on LAN trigger 1 to digital I/O trigger 14, then generate the event ID for 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	System reset LAN trigger N reset	Create configuration script	"0.0.0.0"

Usage

```
ipAddress = lan.trigger[N].ipaddress
lan.trigger[N].ipaddress = ipAddress
```

<i>ipAddress</i>	The LAN address for this attribute as a string in dotted decimal notation
<i>N</i>	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()
```

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 7-116)

lan.trigger[N].mode

This attribute sets the trigger operation and detection mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset LAN trigger N reset	Create configuration script	lan.TRIG_EITHER

Usage

```
mode = lan.trigger[N].mode
lan.trigger[N].mode = mode
```

<i>mode</i>	A number representing the LAN event number (1 to 8); see the Details section for more information
<i>N</i>	A number representing the trigger mode (0 to 7)

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.

mode settings

Full	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

Details

lan.TRIG_RISING and lan.TRIG_RISINGA are the same.

lan.TRIG_RISING and lan.TRIG_RISINGM are the same.

lan.TRIG_SYNCHRONOUS is provided for compatibility with the digital I/O and TSP-Link triggering on older firmware. Use of lan.TRIG_SYNCHRONOUSA or lan.TRIG_SYNCHRONOUSM (instead of lan.TRIG_SYNCHRONOUS) is preferred.

Example

```
print(lan.trigger[1].mode)
```

Returns the present LAN trigger mode of LAN event 1.

Also see

[digio functions and attributes](#) (on page 5-9)

[tsplink functions and attributes](#) (on page 5-16)

lan.trigger[N].overrun

This attribute describes event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
overrun = lan.trigger[N].overrun
```

<i>overrun</i>	The trigger overrun state for the specified LAN packet (<code>true</code> or <code>false</code>)
<i>N</i>	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. Output trigger overrun indications are provided in the status model.

Example

```
overrun = lan.trigger[5].overrun
print(overrun)
```

Checks the overrun status of a trigger with LAN packet 5 and return the value, such as:
false

Also see

[lan.trigger\[N\].assert\(\)](#) (on page 7-114)

[lan.trigger\[N\].clear\(\)](#) (on page 7-115)

[lan.trigger\[N\].stimulus](#) (on page 7-123)

[lan.trigger\[N\].wait\(\)](#) (on page 7-125)

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	System reset LAN trigger N reset	Create configuration script	lan.TCP

Usage

```
protocol = lan.trigger[N].protocol
lan.trigger[N].protocol = protocol
```

<i>protocol</i>	The protocol to use for the trigger's messages: <ul style="list-style-type: none"> 0 (lan.TCP) 1 (lan.UDP) 2 (lan.MULTICAST)
<i>N</i>	The LAN event number (1 through 8)

Details

The LAN trigger listens for trigger messages from either protocol 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 `ipAddress` attribute is ignored and event messages are sent to the multicast address 224.0.23.159.

Example

<code>print(lan.trigger[1].protocol)</code>	Get LAN protocol to use for sending trigger messages for LAN event 1.
---------------------------------------------	-----------------------------------------------------------------------

Also see

[lan.trigger\[N\].connect\(\)](#) (on page 7-116)

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	System reset LAN trigger N reset	Create configuration script	1

Usage

```
pseudostate = lan.trigger[N].pseudostate
lan.trigger[N].pseudostate = pseudostate
```

<i>pseudostate</i>	The simulated line state (0 or 1)
<i>N</i>	The LAN event number (1 to 8)

Details

This attribute can initialize the pseudo state to a known value.
 Setting this attribute will not cause the LAN trigger to generate any events or output packets.
 ON or OFF cannot be used when setting the pseudostate.

Example

<code>print(lan.trigger[1].pseudostate)</code>	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	System reset LAN trigger N reset	Create configuration script	0

Usage

```
triggerStimulus = lan.trigger[N].stimulus
lan.trigger[N].stimulus = triggerStimulus
```

<i>triggerStimulus</i>	The LAN event identifier used to trigger the event
<i>N</i>	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. The events (*triggerStimulus*) may be one of the existing trigger event IDs, shown in the following table.

Trigger event IDs

Trigger event ID	Description
<code>digio.trigger[N].EVENT_ID</code>	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
<code>display.trigger.EVENT_ID</code>	The trigger key on the front panel is pressed
<code>trigger.EVENT_ID</code>	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event
<code>trigger.blender[N].EVENT_ID</code>	A combination of events has occurred
<code>trigger.timer[N].EVENT_ID</code>	A delay expired
<code>tsplink.trigger[N].EVENT_ID</code>	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line
<code>lan.trigger[N].EVENT_ID</code>	A LAN trigger event has occurred
<code>scan.trigger.EVENT_SCAN_READY</code>	Scan Ready Event
<code>scan.trigger.EVENT_SCAN_START</code>	Scan Start Event
<code>scan.trigger.EVENT_CHANNEL_READY</code>	Channel Ready Event
<code>scan.trigger.EVENT_SCAN_COMP</code>	Scan Complete Event
<code>scan.trigger.EVENT_IDLE</code>	Idle Event

NOTE

Use the `EVENT_ID` constant to set the stimulus attribute rather than using a numerical value for the `EVENT_ID` constant. Doing this will make your code compatible for possible upgrades if the `EVENT_ID` changes in future enhancements to the instrument.

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 7-114)
- [lan.trigger\[N\].clear\(\)](#) (on page 7-115)
- [lan.trigger\[N\].connect\(\)](#) (on page 7-116)
- [lan.trigger\[N\].overrun](#) (on page 7-121)
- [lan.trigger\[N\].wait\(\)](#) (on page 7-125)

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)
```

<i>triggered</i>	Trigger detection indication
<i>N</i>	The trigger packet over LAN to wait for (1 to 8)
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger event

Details

If one or more trigger events have been detected since the last time `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 7-114)

[lan.trigger\[N\].clear\(\)](#) (on page 7-115)

[lan.trigger\[N\].overrun](#) (on page 7-121)

[lan.trigger\[N\].stimulus](#) (on page 7-123)

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

```

<i>maxNumber</i>	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 to 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.

Example 1

<pre>maxNumber = localnode.define.MAX_TIMERS</pre>	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 of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes		Nonvolatile memory	Instrument specific (see Details)

Usage

```

localnode.description = description
description = localnode.description

```

<i>description</i>	User-defined description of the instrument
--------------------	--------------------------------------------

Details

This attribute stores a string that contains a description of the instrument. This value appears on instrument's LXI home page.

This attribute's default value contains Keithley *MMMM #SSSSSSSS*, where: *MMMM* is the instrument's four-digit 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.

Example

<pre>description = "System in Lab 05" localnode.description = description</pre>	Set description equal to "System in Lab 05". Set the LXI home page of this instrument to display "System in Lab 05".
---------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------

Also see

[Using the web interface](#) (on page 2-69)

localnode.execute()

This function starts test scripts on a remote node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
node[N].execute(scriptCode)
```

<i>N</i>	The node number of the instrument on which to execute <i>script</i>
<i>scriptCode</i>	A string containing the source code

Details

Only the master node can issue the execute command to a remote node. This function does not run test scripts on the master node, only on a remote node.

This function may only be called when the group number of the node is different than the node of the master.

This function will not wait for the script to finish execution.

This function cannot be used on the local node. It is provided for the sole purpose of executing scripts on a node from a remote master node. The `localnode` prefix to the function listing describes how remote commands are shared between nodes, rather than this command being a `localnode` function.

Example 1

```
node[2].execute(sourcecode)      Runs script code on node 2.
```

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.source" (previously stored on the master  
node) on node 32.
```

Also see

[Introduction to TSP advanced features](#) (on page 6-49)

localnode.getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
value = node[N].getglobal(name)
```

<i>value</i>	The value of the variable
<i>N</i>	The node number of the instrument retrieving the global variable from its runtime environment
<i>name</i>	The global variable name

Details

Use this function to retrieve the value of a global variable from a remote node.

Do not use this command to retrieve the value of a global variable from the local node. It is provided for the sole purpose of accessing global variables on a node from a remote master node. The `localnode` prefix to the function listing describes how remote commands are shared between nodes, rather than this command being a `localnode` command.

Example

```
print(node[5].getglobal("test_val"))
```

Retrieves and outputs the value of the global variable named `test_val` from Node 5.

Also see

[Introduction to TSP advanced features](#) (on page 6-49)

[localnode.setglobal\(\)](#) (on page 7-133)

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
```

<i>model</i>	The model number of the instrument
--------------	------------------------------------

Example

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

Outputs the model number.

Also see

[localnode.description](#) (on page 7-126)

[localnode.revision](#) (on page 7-131)

[localnode.serialno](#) (on page 7-132)

localnode.password

This attribute stores the remote access password.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (W)	Yes	LAN reset LAN restore defaults	Nonvolatile memory	"admin"

Usage

```
localnode.password = password
```

<i>password</i>	String containing the remote interface password
-----------------	-------------------------------------------------

Details

This write-only attribute (cannot be read) stores the password set for any remote interface. When password usage is enabled (`localnode.passwordmode`), supply this 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 cannot retrieve a lost password from any command interface.

The password can be reset by resetting the LAN from the front panel or by assigning an empty string to this attribute.

Example

<code>localnode.password = "N3wpa55w0rd"</code>	Changes the remote interface password to N3wpa55w0rd.
-------------------------------------------------	-------------------------------------------------------

Also see

None

localnode.prompts

This attribute stores local node prompting state (enabled or disabled).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not saved	Disabled

Usage

```
prompting = localnode.prompts
```

```
localnode.prompts = prompting
```

<i>prompting</i>	Prompting state, 0 (disable) or 1 (enable)
------------------	--------------------------------------------

Details

This attribute controls prompting. When set to 1, prompts are issued after each command message is processed by the instrument. When set to 0, prompts are not issued.

The command messages do not generate prompts. The instrument generates prompts in response to command messages.

When the prompting mode is enabled, the instrument generates prompts in response to command messages. There are three prompts that might be returned:

- **TSP>** is the standard prompt. This prompt indicates that everything is normal and the command is done processing.
- **TSP?** is issued if there are entries in the error queue when the prompt is issued. Like the "TSP>" prompt, it indicates the command is done processing. It does not mean the previous command generated an error, only that there are still errors in the queue when the command was done processing.
- **>>>>** is the continuation prompt. This prompt is used when downloading scripts. When downloading scripts, many command messages must be sent as a unit. The continuation prompt indicates that the instrument is expecting more messages as part of the current command.

Test Script Builder requires prompts. It sets the prompting mode behind the scenes. If you disable prompting, use of the Test Script Builder will hang because it will be waiting for the prompt that lets it know that the command is done executing. DO NOT disable prompting with the use of the Test Script Builder.

When used in an expanded system (TSP-Link), `localnode.prompts` is sent to the remote master node only. Use `node[N].prompts` (where *N* is the node number) to send the command to any node in the system.

Example

```
localnode.prompts = 1      Enable prompting.
```

Also see

[localnode.showerrors](#) (on page 7-133)

[localnode.prompts4882](#) (on page 7-130)

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	Not applicable	Not saved	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 generates 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.

Example

```
localnode.prompts4882 = 0
```

Disables IEEE Std. 488.2 common command prompting.

Also see

[localnode.prompts](#) (on page 7-129)

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')` and a `scan.reset()`. In addition:

- Other settings are restored back to factory default settings
- Existing channel patterns are deleted
- All channels are opened

Example

```
localnode.reset()
```

Resets the local node.

Also see

[channel.reset\(\)](#) (on page 7-43)

[reset\(\)](#) (on page 7-140)

[scan.reset\(\)](#) (on page 7-150)

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
```

<code>revision</code>	Firmware revision level
-----------------------	-------------------------

Details

This attribute indicates the firmware revision number currently running in the instrument.

Example

```
print(localnode.revision)
```

Outputs the present revision level. Sample output:
01.00a

Also see

[localnode.description](#) (on page 7-126)

[localnode.model](#) (on page 7-128)

[localnode.serialno](#) (on page 7-132)

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
```

<code>serialno</code>	The serial number of the instrument
-----------------------	-------------------------------------

Details

This read-only attribute indicates the instrument serial number.

Example

```
display.clear()
```

```
display.settext(localnode.serialno)
```

Clears the unit's display.

Places the unit's serial number on the top line of its display.

Also see

[localnode.description](#) (on page 7-126)

[localnode.model](#) (on page 7-128)

[localnode.revision](#) (on page 7-131)

localnode.setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

```
node[N].setglobal(name, value)
```

<i>N</i>	The node number of the instrument
<i>name</i>	The global variable name to create
<i>value</i>	The value to assign to the variable

Details

From a remote node, use this function to create and 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. It is provided for the sole purpose of accessing global variables on a node from a remote master node. The `localnode` prefix to the function listing describes how remote commands are shared between nodes, rather than this command being a `localnode` command.

Example

```
node[3].setglobal("x", 5)      Sets the global variable x on Node 3 to the value of 5.
```

Also see

[localnode.getglobal\(\)](#) (on page 7-128)

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	Disabled

Usage

```
errorMode = localnode.showerrors
localnode.showerrors = errorMode
```

<i>errorMode</i>	Enables (1) or disables (0) 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 used in an expanded system (TSP-Link), `localnode.showerrors` is sent to the remote master node only. Use `node[N].showerrors` (where *N* is the node number) to send the command to any node in the system.

Example

```
localnode.showerrors = 1
```

Enables sending of generated errors.

Also see

[localnode.prompts](#) (on page 7-129)

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)
```

<i>getter</i>	The return value
<i>table</i>	Read-only table where the attribute is located
<i>attributeName</i>	The string 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

```
getRule = makegetter(channel, "connectrule")
...
r = getrule()
```

Creates a getter function called `getRule`.
When `getRule` is called, it returns the value of `connectrule`.

Also see

[makesetter\(\)](#) (on page 7-135)

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)
```

<i>setter</i>	Function that sets the value of the attribute
<i>table</i>	Read-only table where the attribute is located
<i>attributeName</i>	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. Otherwise, the overhead of creating the setter function outweighs the overhead of accessing the attribute directly.

<pre>setRule = makesetter(channel, "connectrule") r = setrule(channel.BREAK_BEFORE_MAKE)</pre>	<p>Creates a setter function called setRule.</p> <p>When setRule is called, it configures the setting for connectrule. In this example, setting connection rule to break-before-make.</p>
------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[makegetter\(\)](#) (on page 7-134)

[channel.connectrule](#) (on page 7-20)

memory.available()

This function reads and returns the amount of memory that is available in the instrument overall and for user scripts, and storing channel patterns

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
memoryAvailable = memory.available()
```

<i>memoryAvailable</i>	<p>Comma-delimited string with percentages for available memory; the format is <code>systemMemory, scriptMemory, patternMemory</code>, where:</p> <ul style="list-style-type: none"> • <code>systemMemory</code>: The percentage of memory available in the instrument • <code>scriptMemory</code>: The percentage of memory available in the instrument to store user scripts • <code>patternMemory</code>: The percentage of memory available in the instrument to store channel patterns
------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Use this function to view the available memory in the overall instrument as well as the memory available for storing user scripts and channel patterns.

The response to this function is a single string that returns the overall instrument memory available, script memory available, and channel pattern memory available as comma-delimited percentages.

Example: Available memory

<pre>memoryAvailable = memory.available() print(memoryAvailable)</pre>	<p>Reads and returns the amount of memory available in the instrument.</p> <p>Output:</p> <pre>51.56, 92.84, 100.00</pre> <p>You can also use:</p> <pre>print(memory.available())</pre>
------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example: Used and available memory

<pre>print("Memory used:", memory.used()) print("Memory available: ", memory.available())</pre>	<p>Reads and returns the amount memory used and memory available percentages.</p> <p>Output:</p> <pre>Memory used: 69.14, 0.16, 12.74 Memory available: 30.86, 99.84, 87.26</pre>
-------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[memory.used\(\)](#) (on page 7-137)

memory.used()

This function reports the amount of memory used in the instrument overall and for user scripts, and storing channel patterns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
memoryUsed = memory.used()
```

<i>memoryUsed</i>	<p>A comma-delimited string with percentages for used memory; the format is <code>systemMemory, scriptMemory, patternMemory</code>, where:</p> <ul style="list-style-type: none"> <code>systemMemory</code>: The percentage of memory used in the instrument <code>scriptMemory</code>: The percentage of memory used in the instrument to store user scripts <code>patternMemory</code>: The percentage of memory used in the instrument to store channel patterns
-------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Use this function to view the used memory in the overall instrument, as well as the memory used for storing user scripts and channel patterns.

The response to this function is a single string that shows the overall instrument memory used, as well as the script memory used and channel pattern memory used as comma-delimited percentages.

Example

<pre>MemUsed = memory.used() print(MemUsed)</pre>	<p>Reads the memory used in the instrument and returns out the percentages. Output: 69.14, 0.16, 12.74</p>
---------------------------------------------------	--------------------------------------------------------------------------------------------------------------------

Also see

[memory.available\(\)](#) (on page 7-135)

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 not actively performing overlapped commands will set their bits immediately. All remaining nodes will set their own bits as they complete their own overlapped commands.

For additional information, see Status Model.

Also see

[waitcomplete\(\)](#) (on page 7-243)

print()

This function returns the value of the argument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

<i>value1</i>	The first argument to return
<i>value2</i>	The second argument to return
<i>valueN</i>	The last argument to return

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.

Example

```
x = 10
print(x)
```

Example of an output response message:

```
1.0000000e+01
```

Note that your output might be different if you set your ASCII precision setting to a different value.

Also see

None

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, buffer1)
printbuffer(startIndex, endIndex, buffer1, buffer2)
printbuffer(startIndex, endIndex, buffer1, ..., bufferN)
```

<i>startIndex</i>	Beginning index of the buffer that is to be printed
<i>endIndex</i>	Ending index of the buffer that is to be printed
<i>buffer1, buffer2, bufferN</i>	Tables or reading buffer subtables that are to be printed

Details

The correct usage of this function for a buffer containing n elements is:

$$1 \leq \text{startIndex} \leq \text{endIndex} \leq n$$

Where n refers to the index of the last entry in the tables to be printed.

If $\text{endIndex} < \text{startIndex}$ or $n < \text{startIndex}$, no data is printed. If $\text{startIndex} \leq 1$, 1 is used as startIndex . If $n < \text{endIndex}$, n is used as endIndex .

When any given reading buffers are being used in overlapped commands that have not yet completed, at least to the desired index, this function returns data as it becomes available.

When there are outstanding overlapped commands to acquire data, n refers to the index that the last entry in the table will have after all the measurements have completed.

If you do not specify a subtable in a reading buffer, default subtables are automatically used.

At least one table or subtable must be specified.

This command generates a single response message that contains all data. The response message is stored in the output queue.

The `format.data` attribute controls the format of the response message.

Example

```
format.data = format.ASCII
format.asciiprecision = 6
printbuffer(1, rbl.n, rbl)
```

This assumes that `rbl` is a valid reading buffer in the runtime environment.

Example of returned data (`rbl.readings`):

```
4.07205e-05, 4.10966e-05, 4.06867e-05, 4.08865e-05, 4.08220e-05, 4.08988e-05,
4.08250e-05, 4.09741e-05, 4.07174e-05, 4.07881e-05
```

Also see

[format.asciiprecision](#) (on page 7-93)

[format.byteorder](#) (on page 7-93)

[format.data](#) (on page 7-94)

[print\(\)](#) (on page 7-138)

[printnumber\(\)](#) (on page 7-140)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value)
printnumber(value, value2, ..., value_N)
```

<code>value, value2, value_N</code>	Numbers to print in the configured format
-------------------------------------	-------------------------------------------

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`, `format.asciiprecision`, and other associated attributes.

At least one number must be given.

Example

<pre>format.asciiprecision = 10 x = 2.54 printnumber(x) format.asciiprecision = 3 printnumber(x, 2.54321, 3.1)</pre>	<p>Output:</p> <pre>2.540000000e+00 2.54e+00, 2.54e+00, 3.10e+00</pre>
----------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------

Also see

[format.asciiprecision](#) (on page 7-93)

[format.byteorder](#) (on page 7-93)

[format.data](#) (on page 7-94)

[print\(\)](#) (on page 7-138)

[printbuffer\(\)](#) (on page 7-138)

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)
```

<code>system</code>	<p>true: If the node is the master, the entire system is reset</p> <p>false: Only the local group is reset</p>
---------------------	----------------------------------------------------------------------------------------------------------------

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. The `localnode.reset()` command is used for the local instrument. The `node[X].reset()` command is used to reset an instrument on a subordinate node.

When no value is specified for *system*, the default value is `true`.

Resetting the entire system using the `reset(true)` function is only permitted if the reset command sent from the controlling node. If the node is not the controller, sending this command generates an error.

Example

<code>reset(true)</code>	If sent from the controlling node, the entire system is reset (if not sent from the controlling node, an error is generated).
--------------------------	-------------------------------------------------------------------------------------------------------------------------------

Also see

[localnode.reset\(\)](#) (on page 7-131)

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 they were in when the scan was aborted.

Example

<code>scan.background()</code> <code>scan.abort()</code>	Starts background scan, and then aborts the scan.
-------------------------------------------------------------	---------------------------------------------------

Also see

[scan.background\(\)](#) (on page 7-144)

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)
```

<i>channelList</i>	String specifying channels to add using normal channel list syntax
--------------------	--------------------------------------------------------------------

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*. Specifying multiple channels in *channelList* adds multiple steps to the scan.

If an error is encountered as channels are added to the list, subsequent channels in that channel list will not be added.

Example 1

<pre>scan.create() for column = 1,5 do scan.add(channel.createspecifier(1,1,column n)) end</pre>	<p>Replaces the active scan list with an empty scan list.</p> <p>Loops through columns 1 to 5.</p> <p>Adds five channels to the scan list using the <code>channel.createspecifier()</code> command. The scan list now has row 1, columns 1 to 5 on slot 1 as the first five steps.</p>
---------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example 2

<pre>scan.create("1A01:1A08") scan.add("1A10") scan.add("1A09")</pre>	<p>Replaces the active scan list with an empty scan list, and then adds row A, columns 1 through 8 on slot 1 to the new scan list.</p> <p>Adds row A, column 10 on slot 1 to the end of the scan list.</p> <p>Adds row A, column 9 on slot 1 to the end of the scan list.</p> <p>Scan list now includes channels 1A01 through 1A10, with channels 1A01 through 1A08 in order, followed by channel 1A10, and then channel 1A09.</p>
-------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Example 3

```
scan.create( " " )
```

Clears the old scan list and creates a new empty scan list.

Also see

[scan.create\(\)](#) (on page 7-146)

[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)
```

```
channelList
```

String specifying a list of channels

Details

This function adds a list of channels to be closed simultaneously in a single step of a scan.

Example

```
scan.create()
scan.add("1D01")
scan.addimagestep("1A01, 1B01, 1C03")
scan.add("1F03")
scan.addimagestep("1A03, 1B03, 1C03")
scan.addimagestep("1A05, 1B05, 1C03")
scan.addimagestep("1A07, 1B07, 1C03")
scan.addimagestep("1A09, 1B09, 1C03")
print(scan.list())
```

Generate a scan list that has multiple steps, with some steps that include multiple channels.

Output:

```
Init) OPEN...
  1) STEP: 1D01
      CLOSE: 1D01
  2) STEP: 1A01, 1B01, 1C03
      OPEN: 1D01
      CLOSE: 1A01 1B01 1C03
  3) STEP: 1F03
      OPEN: 1A01 1B01 1C03
      CLOSE: 1F03
  4) STEP: 1A03, 1B03, 1C03
      OPEN: 1F03
      CLOSE: 1A03 1B03 1C03
  5) STEP: 1A05, 1B05, 1C03
      OPEN: 1A03 1B03
      CLOSE: 1A05 1B05
  6) STEP: 1A07, 1B07, 1C03
      OPEN: 1A05 1B05
      CLOSE: 1A07 1B07
  7) STEP: 1A09, 1B09, 1C03
      OPEN: 1A07 1B07
      CLOSE: 1A09 1B09
```

Also see

[scan.add\(\)](#) (on page 7-142)

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 = scan.background()
```

<i>state</i>	The result of scanning: <code>scan.EMPTY</code> or 0 <code>scan.BUILDING</code> or 1 <code>scan.RUNNING</code> or 2 <code>scan.ABORTED</code> or 3 <code>scan.FAILED</code> or 4 <code>scan.FAILED_INIT</code> or 5 <code>scan.SUCCESS</code> or 6
<i>scanCount</i>	The present number of scans completed
<i>stepCount</i>	The present number of steps completed

Details

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

<code>scan.background()</code>	Runs a scan in the background
--------------------------------	-------------------------------

Also see

[scan.add\(\)](#) (on page 7-142)

[scan.create\(\)](#) (on page 7-146)

[scan.execute\(\)](#) (on page 7-147)

[scan.list\(\)](#) (on page 7-148)

[scan.state\(\)](#) (on page 7-152)

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	System reset Scan reset	Create configuration script	Scan.ON

Usage

```
bypass = scan.bypass
```

```
scan.bypass = bypass
```

<i>bypass</i>	The state of the bypass. Set to one of the following values: <code>scan.OFF</code> or 0: Bypass disabled <code>scan.ON</code> or 1: Bypass enabled
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------------

Details

When *bypass* is ON and the `scan.trigger.arm.stimulus` is satisfied, 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.

Also see

[scan.trigger.arm.stimulus](#) (on page 7-154)

[scan.trigger.channel.stimulus](#) (on page 7-157)

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)
```

channelList

String specifying channels to add

Details

The existing scan list is lost after calling this function.

The items in *channelList* are scanned in the order listed.

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.

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("1A01:1A10")
```

Replaces the active scan list with an empty scan list, and then adds channels 1A01 through 1A10 on slot 1.

Example 2

```

scan.create()

for column = 1, 10 do

    scan.add(channel.createspecifier(1,1,column
n))

end
    
```

Replaces the active scan list with an empty scan list.

Loops through column 1 to 10 on row 1 of slot 1 to add ten channels to the scan list. The `channel.createspecifier()` command generates the parameters.

The scan list now has, in order, row 1, columns 1 through 10, on slot 1.

Also see

- [scan.add\(\)](#) (on page 7-142)
- [scan.reset\(\)](#) (on page 7-150)

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

```
scanState, scanCount, stepCount = scan.execute()
```

<i>scanState</i>	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
<i>scanCount</i>	The current number of scans that have completed
<i>stepCount</i>	The current number of steps have completed

Details

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.

Example

```
scan.execute()
```

Runs a scan immediately.

Also see

[scan.add\(\)](#) (on page 7-142)

[scan.background\(\)](#) (on page 7-144)

[scan.create\(\)](#) (on page 7-146)

[scan.list\(\)](#) (on page 7-148)

[scan.state\(\)](#) (on page 7-152)

scan.list()

This function queries the active scan list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes	System reset Channel reset Changing a channel or scan setting	Create configuration script	Empty list

Usage

```
scanList = scan.list()
```

```
scanList
```

This parameter is string listing the existing scan step information

Details

This function lists the existing scan list.

When changing a channel or scan attribute for an existing scan list item, the scan list will be regenerated based on this change. If unable to rebuild the list, an error will be generated and the scan list will be lost. To avoid unintentional changes to an existing scan list, configure channel and scan settings prior to using commands (`scan.add()`, `scann.addimagestep()`, `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, and close (see the example below).

Example

```
scan.create("1A07:1B03")  
print(scan.list())
```

Populate the scan list with the function `scan.create("1A07:1B03")`, then initiate the scan list to be output.

Outputs the existing scan list.

Output:

```
Init) OPEN...  
  1) STEP: 1A07  
     CLOSE: 1A07  
  2) STEP: 1A08  
     OPEN: 1A07  
     CLOSE: 1A08  
  3) STEP: 1A09  
     OPEN: 1A08  
     CLOSE: 1A09  
  4) STEP: 1A10  
     OPEN: 1A09  
     CLOSE: 1A10  
  5) STEP: 1A11  
     OPEN: 1A10  
     CLOSE: 1A11  
  6) STEP: 1A12  
     OPEN: 1A11  
     CLOSE: 1A12  
  7) STEP: 1B01  
     OPEN: 1A12  
     CLOSE: 1B01  
  8) STEP: 1B02  
     OPEN: 1B01  
     CLOSE: 1B02  
  9) STEP: 1B03  
     OPEN: 1B02  
     CLOSE: 1B03
```

Also see

[scan.create\(\)](#) (on page 7-146)

scan.mode

This attribute controls the scan mode setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Scan reset	Create configuration script	scan.MODE_OPEN_ALL

Usage

```
scanModeSetting = scan.mode
scan.mode = scanModeSetting
```

<i>scanModeSetting</i>	The present scan mode setting. Set to one of the following values: <ul style="list-style-type: none"> scan.MODE_OPEN_ALL or 0: Opens all relays on all slots before a scan starts scan.MODE_OPEN_SELECTIVE or 1: Opens relays intelligently; 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:

- All channels involved in scanning are opened
- Closed channels not involved in scanning remain closed during the scan

Example

<code>scan.mode = scan.MODE_OPEN_SELECTIVE</code>	Sets the scan mode setting to open selective.
---------------------------------------------------	-----------------------------------------------

Also see

[scan.reset\(\)](#) (on page 7-150)

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.mode`

- `scan.scancount`
- `scan.trigger.arm.stimulus`
- `scan.trigger.channel.stimulus`

In addition, the scan list is cleared.

NOTE

Sending this function only affects the trigger model and scan list settings. To reset the entire system 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 7-43)

[reset\(\)](#) (on page 7-140)

[scan.bypass](#) (on page 7-145)

[scan.mode](#) (on page 7-150)

[scan.scancount](#) (on page 7-151)

[scan.trigger.arm.stimulus](#) (on page 7-154)

[scan.trigger.channel.stimulus](#) (on page 7-157)

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	System reset Scan reset	Create configuration script	1

Usage

```
scanCount = scan.scancount
```

```
scan.scancount = scanCount
```

```
scanCount
```

The present scan count value (1 to 2,000,000,000)

Details

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 3-1)

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 = scan.state()
```

<i>scanState</i>	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
<i>scanCount</i>	The current number scans that have completed
<i>stepCount</i>	The current number steps that have 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 yet to begin, 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:

```
2.0000000000000000e+00
```

Also see[scan.mode](#) (on page 7-150)[scan.background\(\)](#) (on page 7-144)

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
```

<code>scanStepCount</code>	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

<pre>print(scan.stepcount)</pre>	Responds with the present step count. Output assuming there are five steps in the scan list: 5.0000000000000000e+00
----------------------------------	---------------------------------------------------------------------------------------------------------------------------

Also see[scan.add\(\)](#) (on page 7-142)[scan.create\(\)](#) (on page 7-146)[scan.addimagestep\(\)](#) (on page 7-143)

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

<code>scan.trigger.arm.clear()</code>	Clears the arm event detector.
---------------------------------------	--------------------------------

Also see

[scan.trigger.arm.set\(\)](#) (on page 7-154)

[scan.trigger.arm.stimulus](#) (on page 7-154)

[Trigger model](#) (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

<code>scan.trigger.arm.set()</code>	Sets the arm event detector to the detected state.
-------------------------------------	----------------------------------------------------

Also see

[scan.trigger.arm.clear\(\)](#) (on page 7-153)

[scan.trigger.arm.stimulus](#) (on page 7-154)

[Trigger model](#) (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	System reset Scan reset	Create configuration script	0

Usage

```
eventID = scan.trigger.arm.stimulus
scan.trigger.arm.stimulus = eventID
```

<code>eventID</code>	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:

- `digio.trigger[N].EVENT_ID`: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
- `display.trigger.EVENT_ID`: The trigger key on the front panel is pressed
- `trigger.EVENT_ID`: A *trg message on the active command interface. If GPIB is the active command interface, a GET message will also generate this event
- `trigger.blender[N].EVENT_ID`: A combination of configured events has occurred
- `trigger.timer[N].EVENT_ID`: A delay expired
- `tsplink.trigger[N].EVENT_ID`: An edge (either rising, falling, or either based on the configuration of the line) on the tsplink trigger line
- `lan.trigger[N].EVENT_ID`: Event identifier use to route the LAN trigger to other subsystems (using stimulus properties)
- `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_SCAN_COMP`: Scan Complete Event
- `scan.trigger.EVENT_IDLE`: Idle Event

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 7-153)

[scan.trigger.arm.set\(\)](#) (on page 7-154)

[Trigger model](#) (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

<code>scan.trigger.channel.clear()</code>	Clears the channel event detector.
-------------------------------------------	------------------------------------

Also see

[scan.trigger.channel.set\(\)](#) (on page 7-156)
[scan.trigger.channel.stimulus](#) (on page 7-157)
[Trigger model](#) (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

<code>scan.trigger.channel.set()</code>	Sets the channel event detector of the trigger model to the detected state.
-----------------------------------------	-----------------------------------------------------------------------------

Also see

[scan.trigger.channel.clear\(\)](#) (on page 7-156)
[scan.trigger.channel.stimulus](#) (on page 7-157)
[Trigger model](#) (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	System reset Scan reset	Create configuration script	scan.trigger.EVENT_CHANNEL_READY

Usage

```
eventID = scan.trigger.channel.stimulus
scan.trigger.channel.stimulus = eventID
```

<i>eventID</i>	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.

The *eventID* parameter may be one of the following trigger event IDs:

- `digio.trigger[N].EVENT_ID`: An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
- `display.trigger.EVENT_ID`: The trigger key on the front panel is pressed
- `trigger.EVENT_ID`: A *trg message on the active command interface; if GPIB is the active command interface, a GET message will also generate this event
- `trigger.blender[N].EVENT_ID`: A combination of configured events has occurred
- `trigger.timer[N].EVENT_ID`: A delay expired
- `tsplink.trigger[N].EVENT_ID`: An edge (either rising, falling, or both, depending upon the configuration of the line) on the tsplink trigger line
-
- `lan.trigger[N].EVENT_ID`: Event identifier used to route the LAN trigger to other subsystems (using stimulus properties)
- `scan.trigger.EVENT_SCAN_START`: Scan start event
- `scan.trigger.EVENT_CHANNEL_READY`: Channel ready event
- `scan.trigger.EVENT_SCAN_COMP`: Scan complete event
- `scan.trigger.EVENT_IDLE`: Idle event

NOTE

Use one of the existing 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 when enhancements are added to the instrument.

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
```

Starts the channel action immediately after the Scan Start Event.

Also see

[scan.trigger.channel.clear\(\)](#) (on page 7-156)

[scan.trigger.channel.set\(\)](#) (on page 7-156)

[Trigger model](#) (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 and channel 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 7-156)

[scan.trigger.channel.stimulus](#) (on page 7-157)

[Trigger model](#) (on page 3-1)

script.anonymous

This is a reference to the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	No	See Details	See Details	Not applicable

Usage

```
scriptVar = script.anonymous
```

Details

You can use the `script.anonymous` script like any other script. Also, you can save the anonymous script as a user script provided it is given a name.

This script is replaced by the commands `loadscript` and `loadandrunscript` 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 6-4)

[scriptVar.autorun](#) (on page 7-163)

[scriptVar.list\(\)](#) (on page 7-164)

[scriptVar.name](#) (on page 7-164)

[scriptVar.run\(\)](#) (on page 7-166)

[scriptVar.save\(\)](#) (on page 7-166)

[scriptVar.source](#) (on page 7-167)

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 6-13)

[Delete user scripts from the instrument](#) (on page 6-43)

[scriptVar.save\(\)](#) (on page 7-166)

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)
```

<i>scriptVar</i>	The name of the variable that will reference the script
<i>code</i>	A string containing the body of the script
<i>name</i>	The name of the script

Details

The *name* attribute 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 *name* attribute of the existing script is set to an empty string before it is replaced by the new script. Also, if a name is not given, an empty string is used for the name.

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: Create new script

```
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: Create new autoexec script

```
autoexec = script.new(
    "display.clear() display.setText('Hello from autoexec')", 'autoexec')
Creates a new 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 6-38)
- [Global variables and the script.user.scripts table](#) (on page 6-36)
- [Named scripts](#) (on page 6-4)
- [scriptVar.save\(\)](#) (on page 7-166)
- [script.newautorun\(\)](#) (on page 7-161)

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)
```

<i>scriptVar</i>	The name of the variable that will reference the script
<i>code</i>	A string containing the body of the script
<i>name</i>	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".

Also see

- [Create a script using the script.new command](#) (on page 6-38)
- [Global variables and the script.user.scripts table](#) (on page 6-36)
- [Named scripts](#) (on page 6-4)
- [scriptVar.save\(\)](#) (on page 7-166)
- [script.new\(\)](#) (on page 7-160)

script.restore()

This function restores a script that was removed from the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
script.restore(name)
```

<i>name</i>	The name of the script to be restored
-------------	---------------------------------------

Details

This command copies the script from nonvolatile memory back into the runtime environment, and it 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 7-159)

script.run()

This function runs the anonymous script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

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

Example

```
run()
```

Runs the anonymous script.

Also see

None

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
```

<i>name</i>	String representing the user-defined name of the channel pattern
<i><body></i>	Code that implements the body of the <code>for</code> loop to process the names in the catalog

Example

<pre>for name in script.user.catalog() do print(name) end</pre>	Retrieve the catalog listing for user scripts.
-------------------------------------------------------------------	------------------------------------------------

Also see

None

scriptVar.autorun

This attribute sets a script to autorun.

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
```

<i>scriptVar</i>	The name of the variable that references the script
<i>state</i>	Whether or not the script runs automatically when powered on: <ul style="list-style-type: none"> "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

Save the script in nonvolatile memory after setting the `autorun` attribute to save the change when the instrument is turned off.

Example

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

Assume a script named "test5" is in the runtime 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()
```

<code>scriptVar</code>	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

```
beep.list()
```

Generates output for the content of the script named "beep."

Also see

[Create a script by sending commands over the remote interface](#) (on page 6-5)

[Retrieve source code line by line](#) (on page 6-10)

scriptVar.name

This attribute changes or reads the name of a script in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	See Details	Not applicable

Usage

```
scriptVar.name = scriptName
scriptName = scriptVar.name
```

<i>scriptVar</i>	Name of the variable that references the script
<i>scriptName</i>	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 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: Create a new script with no name and then rename it

```
test7 = script.new("display.clear() display.settext('Hello from my test')", "")
test7()
print(test7.name)

test7.name = "test7"
print(test7.name)

test7.save()
```

Use `script.new()` to create a script with no name, and then run the script. Name the script "test7", and then save the script in nonvolatile memory.

Also see

[script.new\(\)](#) (on page 7-160)

[scriptVar.save\(\)](#) (on page 7-166)

[Rename a script](#) (on page 6-41)

scriptVar.run()

This function runs a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.run()
scriptVar()
```

<i>scriptVar</i>	The name of variable that references the script
------------------	-------------------------------------------------

Example

<code>test8.run()</code>	Runs the script referenced by the variable <code>test8</code> .
--------------------------	-----------------------------------------------------------------

Also see

None

scriptVar.save()

This function saves the script to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()
```

<i>scriptVar</i>	The name of variable that references the script
------------------	-------------------------------------------------

Example

<code>test8.save()</code>	Saves the script referenced by the variable <code>test8</code> to nonvolatile memory.
---------------------------	---------------------------------------------------------------------------------------

Also see

None

scriptVar.source

This attribute holds the source code of a user script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
```

<i>scriptVar</i>	The name of the variable that references the script that contains the source code
<i>code</i>	The body of the script

Details

The `loadscript` or `loadandrunscript` and `endscript` keywords are not included in the source code.

Example

<pre>print(test1.source)</pre>	Gets source code for a script named test1.
--------------------------------	--------------------------------------------

Also see

[scriptVar.list\(\)](#) (on page 7-164)

settime()

This function sets the real-time clock (sets current time of the system).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = settime(time)
result = settime(hour, minute, second)
```

<i>result</i>	A string representing the current time of day
<i>time</i>	The time in seconds since January 1, 1970 UTC. Time to be passed in as: <code>os.time({year = year, month = month, day = day, hour = hour, min = min, sec = sec})</code>
<i>hour</i>	The desired hour from 00 to 23
<i>minute</i>	The desired minute from 00 to 59
<i>second</i>	The desired second from 00 to 59
<i>year</i>	A full year (2006 or later)
<i>month</i>	The desired month from 1 to 12
<i>day</i>	The desired day from 1 to 31

Details

This function sets the date and time of the system based on the `os.time()` response passed in as its parameter. Use year, month, day, hour, min, and sec to set the time as desired. The first three parameters to `os.time()` are mandatory while the rest are optional. If the later 3 are not used, they default to noon for that day. The setting of the time and date does not take into account the time zone. Please update the time for your time zone.

Time must be specified as UTC time. If only the time needs to be changed, the new hour, minutes, and seconds can be passed as arguments without using `os.time()`.

Set the time zone before reading the time using `os.time()` or before generating a UTC time from a local time specification.

Example

<pre>systemTime = os.time({year = 2010, month = 3, day = 31, hour = 14, min = 25}) settime(systemTime) print(settime()) print(settime(10, 10, 10))</pre>	<p>Sets the date and time to Mar 31, 2010 at 2:25 pm.</p> <p>Output:</p> <pre>Wed Mar 31 14:25:05 2010 Wed Mar 31 10:10:10 2010</pre>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------

Also see

[gettimezone\(\)](#) (on page 7-96)

[settimezone\(\)](#) (on page 7-168)

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)
```

<i>offset</i>	String representing offset from UTC
<i>dstOffset</i>	String representing daylight savings offset from UTC
<i>dstStart</i>	String representing when daylight savings time starts
<i>dstEnd</i>	String representing when daylight savings time ends

Details

The time zone is only used when converting between local time and UTC time when using the `os.time()` and `os.date()` functions. The instrument does not track daylight savings time.

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 minutes and seconds fields are optional.

The UTC-5 time zone would be specified with the string "5" because UTC-5 is 5 hours behind UTC and one must add 5 hours to the local time to get UTC time. The time zone UTC4 would be specified as "-4" because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to get 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 within 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

<pre>settimezone(8, 1, "3.3.0/02", "11.2.0/02")</pre>	Sets <code>offset</code> 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.
<pre>settimezone(offset)</pre>	Sets local time zone to <code>offset</code> .

Also see

[gettimezone\(\)](#) (on page 7-96)

[settime\(\)](#) (on page 7-167)

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
```

<i>idnString</i>	The return string
<i>X</i>	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[1].idn)
```

If a Model 7173 card is installed in slot 1, the response is:
7173,4x12 Hi Freq Matrix AAAA,02.01a,99999999

Also see

[slot\[X\] attributes](#) (on page 5-15)

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
```

<i>fourPole</i>	The return value
<i>X</i>	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 7-171)

[slot\[X\].poles.two](#) (on page 7-171)

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
```

<i>onePole</i>	The return value
<i>X</i>	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 7-170)

[slot\[X\].poles.two](#) (on page 7-171)

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
```

<i>twoPole</i>	The return value
<i>X</i>	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 7-171)

[slot\[X\].poles.four](#) (on page 7-170)

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	Nonvolatile memory	0

Usage

```
pseudoCard = slot[X].pseudocard
slot[X].pseudocard = pseudoCard
```

<i>pseudoCard</i>	The pseudocards available for Models 707B and 708B are: <ul style="list-style-type: none"> • <code>slot.PSEUDO_NONE</code> or 0 for no pseudocard selection • 7072 for the Model 7072 8x12 Semiconductor Matrix Card • 70721 for the Model 7072-HV 8x12 High-Voltage Semiconductor Matrix Card • 7173 for the Model 7173-50 8x12 High-Frequency 2-Pole 4x12 Matrix Card • 7174 for the Model 7174A 8x12 Low-Current, High-Speed Matrix Card
<i>X</i>	Slot number (1 to 6)

Details

This attribute only exists for a slot if that slot has no card installed in it. Trying to use this attribute for a slot with an installed card generates an error when writing and a `nil` response when reading.

After assigning a pseudocard, the valid commands and attributes based on that pseudocard now exist for that slot. For example, the `slot[X].idn` attribute is valid.

Changing the pseudocard card assignment from a card to `slot.NONE` invalidates existing scan lists.

Example 1

```
myPseudoCard = slot[3].pseudocard
if myPseudoCard == 7072 then
  print("Pseudo-7072 in Slot #3")
end
```

If Slot 3 is configured to have a 7072 pseudocard, the following message is output:

```
Pseudo-7072 in Slot #3
```

Example 2

```
slot[1].pseudocard = 0
print(slot[1].idn)
slot[1].pseudocard = 7072
print(slot[1].idn)
```

This example requires an empty slot. The slot is set to empty and then set to a valid value.

```
Output:
Empty Slot
7072,Pseudo 8x12
SemiMatrix,00.00a,????????
```

```
slot[1].pseudocard = 0
print(slot[1].idn)
slot[1].pseudocard = 7070
print(slot[1].idn)
```

To change the pseudocard, set the slot to empty again, then define the new card.

```
Output:
Empty Slot
7070,Universal Adapter
Card,00.00a,????????
```

Also see

- [slot\[X\] attributes](#) (on page 5-15)
- [slot\[X\].idn](#) (on page 7-169)

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
```

<i>statusByte</i>	The byte condition register's status returned as a decimal. A zero (0) indicates no bits set. Other decimal values indicate various bit settings.
-------------------	---------------------------------------------------------------------------------------------------------------------------------------------------

Details

This read-only attribute's value is returned as a decimal 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, if a value of 1.29000e+02 (which is 129) is read as the value of the condition register, the binary equivalent is 10000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

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 following table contains descriptions of the bits:

Bit	Value	Description
B0	<code>status.MEASUREMENT_SUMMARY_BIT</code> <code>status.MSB</code>	Set summary bit indicates that an enabled measurement event has occurred. Bit 0 decimal value: 1
B1	<code>status.SYSTEM_SUMMARY_BIT</code> <code>status.SSB</code>	Set summary bit indicates that an enabled system event has occurred. Bit 1 decimal value: 2
B2	<code>status.ERROR_AVAILABLE</code> <code>status.EAV</code>	Set summary bit indicates that an error or status message is present in the Error Queue. Bit 2 decimal value: 4
B3	<code>status.QUESTIONABLE_SUMMARY_BIT</code> <code>status.QSB</code>	Set summary bit indicates that an enabled questionable event has occurred. Bit 3 decimal value: 8
B4	<code>status.MESSAGE_AVAILABLE</code> <code>status.MAV</code>	Set summary bit indicates that a response message is present in the Output Queue. Bit 4 decimal value: 16
B5	<code>status.EVENT_SUMMARY_BIT</code> <code>status.ESB</code>	Set summary bit indicates that an enabled standard event has occurred. Bit 5 decimal value: 32
B6	<code>status.RQS</code> <code>status.MASTER_SUMMARY_STATUS</code> <code>status.MSS</code>	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 or VXI-11 serial poll sequence of the 707B to obtain the status byte (serial poll byte), B6 is the RQS bit. 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 <code>status.condition ICL</code> or the <code>*STB?</code> 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 6 decimal value: 64
B7	<code>status.OPERATION_SUMMARY</code> <code>status.OSB</code>	Set summary bit indicates that an enabled operation event has occurred. Bit 7 decimal value: 128

In addition to the above constants, when more than one bit of the register is set, `statusByte` will equal 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 C-15)

status.node_enable

This attribute stores the status node enable register.

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 node enable register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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, if a value of 1.29000e+02 (which is 129) is read as the value of the node enable register, the binary equivalent is 10000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

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.

The individual bits of the status byte node enable register can be set using constants (for example, to set the enable bit of the standard event status register to B0, set `status.node_enable = status.MSB`). The following table contains descriptions of the bits and available values:

Bit	Value	Description
B0	<code>status.MEASUREMENT_SUMMARY_BIT</code> <code>status.MSB</code>	Set summary bit indicates that an enabled measurement event has occurred. Bit 0 decimal value: 1
B1	Not used	Not applicable.

Bit	Value	Description
B2	<code>status.ERROR_AVAILABLE</code> <code>status.EAV</code>	Set summary bit indicates that an error or status message is present in the Error Queue. Bit 2 decimal value: 4
B3	<code>status.QUESTIONABLE_SUMMARY_BIT</code> <code>status.QSB</code>	Set summary bit indicates that an enabled questionable event has occurred. Bit 3 decimal value: 8
B4	<code>status.MESSAGE_AVAILABLE</code> <code>status.MAV</code>	Set summary bit indicates that a response message is present in the Output Queue. Bit 4 decimal value: 16
B5	<code>status.EVENT_SUMMARY_BIT</code> <code>status.ESB</code>	Set summary bit indicates that an enabled standard event has occurred. Bit 5 decimal value: 32
B6	<code>status.MASTER_SUMMARY_STATUS</code> <code>status.MSS</code>	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set. Bit 6 decimal value: 64
B7	<code>status.OPERATION_SUMMARY</code> <code>status.OSB</code>	Set summary bit indicates that an enabled operation event has occurred. Bit 7 decimal value: 128

In addition to the above values, `nodeEnableRegister` can be set to the decimal 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 status 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 status node enable register using a decimal value.

Also see

[status.condition](#) (on page 7-173)

[status.system.*](#) (on page 7-190)

[Status byte and service request \(SRQ\)](#) (on page C-15)

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
```

<i>nodeEventRegister</i>	The node event register's status in decimal form. A zero (0) indicates no bits set. Other decimal values indicate various bit settings.
--------------------------	-----------------------------------------------------------------------------------------------------------------------------------------

Details

This read-only attribute's value is returned as a decimal 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, if a value of $1.29000e+02$ (which is 129) is read as the value of the condition register, the binary equivalent is 10000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

The returned value can indicate one or more status events occurred.

The following table contains descriptions of the bits:

Bit	Value	Description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB	Set summary bit indicates that an enabled measurement event has occurred. Bit 0 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 2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB	Set summary bit indicates that an enabled questionable event has occurred. Bit 3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV	Set summary bit indicates that a response message is present in the Output Queue. Bit 4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB	Set summary bit indicates that an enabled standard event has occurred. Bit 5 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 6 decimal value: 64
B7	status.OPERATION_SUMMARY status.OSB	Set summary bit indicates that an enabled operation event has occurred. Bit 7 decimal value: 128

In addition to the above constants, *nodeEventRegister* can be set to the decimal equivalent of the bit(s) 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.condition](#) (on page 7-173)

[status.system.*](#) (on page 7-190)

[Status byte and service request \(SRQ\)](#) (on page C-15)

status.operation.*

These attributes manage the status model's operation 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	22528 (All bits set)

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 operation event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

These attribute's values are set using a constant or a decimal value, but are returned as a decimal 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 1.02400e+04 (which is 10,240) is read as the value of the condition register, the binary equivalent is 0010 1000 0000 0000. This value indicates that bit B11 and bit B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, the corresponding bit is set in the event register.

The individual bits of the status byte condition register can be set to the following described values (for example, to set bit B0, set `status.operation.enable = status.MSB`):

Bit	Value	Description
B0-B10	Not used	Not applicable
B11	<code>status.PROMPTS</code> <code>status.PRMPPTS</code>	Set bit indicates that command prompts are enabled. Bit 11 decimal value: 2,048
B12	<code>status.USER</code>	Set bit indicates that an enabled bit in the <code>status.operation.user</code> register is set. Bit 12 decimal value: 4,096
B13	Not used.	Not applicable.
B14	<code>status.PROGRAM_RUNNING</code> <code>status.PROG</code>	Set bit indicates that a program is running. Bit 14 decimal value: 16,384
B15	Not used	Not applicable

In addition to the above constants, `operationRegister` can be set to the decimal 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 2048 + 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	1024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

```
operationRegister = status.PRMPPTS + status.PROG
status.operation.enable = operationRegister
```

Sets the PRMPPTS and PROG bits of the operation status enable register register using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Sets the PRMPTS and PROG bits of the operation status enable register using a decimal value.

Also see

[Operation summary bit \(Operation event register\)](#) (on page C-11)

status.operation.user.*

These attributes manage the status model's operation user register set.

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	32767 (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
```

<i>operationRegister</i>	The user operation event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.
--------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

These attribute's values are set using a constant or a decimal value, but are returned as a decimal 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 10000001. 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

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, a the corresponding bit is set in the event register.

The individual bits of the status byte condition register can be set to the following described values (for example, to set bit B0 of the user operation status register, set `status.operation.user.enable = status.operation.user.BIT0`):

Bit	Value	Description
B0	<code>status.operation.user.BIT0</code>	Bit 0 decimal value: 1
B1	<code>status.operation.user.BIT1</code>	Bit 1 decimal value: 2
B2	<code>status.operation.user.BIT2</code>	Bit 2 decimal value: 4
B3	<code>status.operation.user.BIT3</code>	Bit 3 decimal value: 8
B4	<code>status.operation.user.BIT4</code>	Bit 4 decimal value: 16
B5	<code>status.operation.user.BIT5</code>	Bit 5 decimal value: 32
B6	<code>status.operation.user.BIT6</code>	Bit 6 decimal value: 64
B7	<code>status.operation.user.BIT7</code>	Bit 7 decimal value: 128
B8	<code>status.operation.user.BIT8</code>	Bit 8 decimal value: 256
B9	<code>status.operation.user.BIT9</code>	Bit 9 decimal value: 512
B10	<code>status.operation.user.BIT10</code>	Bit 10 decimal value: 1024
B11	<code>status.operation.user.BIT11</code>	Bit 11 decimal value: 2048
B12	<code>status.operation.user.BIT12</code>	Bit 12 decimal value: 4096
B13	<code>status.operation.user.BIT13</code>	Bit 13 decimal value: 8192
B14	<code>status.operation.user.BIT14</code>	Bit 14 decimal value: 16384
B15	Not used	Not applicable

In addition to the above constants, `operationRegister` can be set to the decimal 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 2048 + 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	1024	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
```

Sets BIT11 and BIT14 of the operation status enable register register using constants.

Example 2

```
-- 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Sets BIT11 and BIT14 of the operation status enable register register using constants using a decimal value.

Also see

[status.operation.*](#) (on page 7-178)

[Operation user bit \(Operation user register\)](#) (on page C-11)

status.questionable.*

These attributes manage the status model's questionable 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	32256 (All bits set)

Usage

```
quesRegister = status.questionable.condition
quesRegister = status.questionable.enable
quesRegister = status.questionable.event
quesRegister = status.questionable.ntr
quesRegister = status.questionable.ptr
```

```
status.questionable.enable = quesRegister
status.questionable.ntr = quesRegister
status.questionable.ptr = quesRegister
```

<i>quesRegister</i>	The questionable event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.
---------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

This attribute's values are set using a constant or a decimal value, but are returned as a decimal 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 1.02400e+04 (which is 10,240) is read as the value of the condition register, the binary equivalent is 0010 1000 0000 0000. This value indicates that bit B11 and bit B13 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit

** Most significant bit

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, the corresponding bit is set in the event register. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B9 of the system summary status register questionable, set `status.questionable.enable = status.questionable.S1THR`):

Bit	Value	Description
B0 to B8	Not used	Not available
B9	<code>status.questionable.S1THR</code> <code>status.questionable.SLOT1_THERMAL</code>	Bit 9 decimal value: 512
B10	<code>status.questionable.S2THR</code> <code>status.questionable.SLOT2_THERMAL</code>	Bit 10 decimal value: 1,024
B11	<code>status.questionable.S3THR</code> <code>status.questionable.SLOT3_THERMAL</code>	Bit 11 decimal value: 2,048
B12	<code>status.questionable.S4THR</code> <code>status.questionable.SLOT4_THERMAL</code>	Bit 12 decimal value: 4,096
B13	<code>status.questionable.S5THR</code> <code>status.questionable.SLOT5_THERMAL</code>	Bit 13 decimal value: 8,192
B14	<code>status.questionable.S6THR</code> <code>status.questionable.SLOT6_THERMAL</code>	Bit 14 decimal value: 16,384
B15	Not used	Not available

In addition to the above constants, `quesRegister` can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set `quesRegister` to the sum of their decimal weights. For example, to set bits B11 and B14, set `quesRegister` to 18,432 (which is the sum of 2048 + 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	1024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

Example 1

<pre>quesRegister = status.questionable.S1THR + status.questionable.S3THR status.questionable.enable = quesRegister</pre>	<p>Sets BIT9 and BIT11 of the status questionable enable register using constants.</p>
---------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------

Example 2

```
-- decimal 2560 = binary 00001010 0000 0000
quesRegister = 2560
status.questionable.enable = quesRegister
```

Sets BIT9 and BIT11 of the status questionable enable register using a decimal value.

Also see

[Questionable summary bit \(Questionable event register\)](#) (on page C-7)

status.request_enable

This attribute stores the status 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
```

<i>requestSRQEnableRegister</i>	The request SRQ enable register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.
---------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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, if a value of $1.29000e+02$ (which is 129) is read as the value of the condition register, the binary equivalent is 10000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

Assigning a value to this attribute enables one or more status events for service request. When an enabled status event occurs, bit B6 of the status byte sets to generate an SRQ (service request).

The individual bits of the status byte condition register can be set using constants (for example, to set the enable bit of the status register to B0, set `status.request_enable = status.MSB`). The following table contains descriptions of the bits and available values:

Bit	Value	Description
B0	<code>status.MEASUREMENT_SUMMARY_BIT</code> <code>status.MSB</code>	Set summary bit indicates that an enabled measurement event has occurred. Bit 0 decimal value: 1
B1	<code>status.SYSTEM_SUMMARY_BIT</code> <code>status.SSB</code>	Set summary bit indicates that an enabled system event has occurred. Bit 1 decimal value: 2
B2	<code>status.ERROR_AVAILABLE</code> <code>status.EAV</code>	Set summary bit indicates that an error or status message is present in the Error Queue. Bit 2 decimal value: 4
B3	<code>status.QUESTIONABLE_SUMMARY_BIT</code> <code>status.QSB</code>	Set summary bit indicates that an enabled questionable event has occurred. Bit 3 decimal value: 8
B4	<code>status.MESSAGE_AVAILABLE</code> <code>status.MAV</code>	Set summary bit indicates that a response message is present in the Output Queue. Bit 4 decimal value: 16
B5	<code>status.EVENT_SUMMARY_BIT</code> <code>status.ESB</code>	Set summary bit indicates that an enabled standard event has occurred. Bit 5 decimal value: 32
B6	Not used.	Not applicable.
B7	<code>status.OPERATION_SUMMARY</code> <code>status.OSB</code>	Set summary bit indicates that an enabled operation event has occurred. Bit 7 decimal value: 128

In addition to the above values, `requestSRQEnableRegister` can be set to the decimal 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
```

Sets the MSB and OSB bits of the request SRQ enable register using constants.

Example 2

```
-- decimal 129 = binary 10000001
requestSRQEnableRegister = 129
status.request_enable = requestSRQEnableRegister
```

Sets the MSB and OSB bits of the request SRQ enable register using a decimal value.

Also see

- [status.condition](#) (on page 7-173)
- [status.system.*](#) (on page 7-190)
- [Status byte and service request \(SRQ\)](#) (on page C-15)

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
```

<i>requestSRQEventRegister</i>	The request event register's status in decimal form. A zero (0) indicates no bits set. Other decimal values indicate various bit settings.
--------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------

Details

This read-only attribute's value is returned as a decimal 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, if a value of $1.29000e+02$ (which is 129) is read as the value of the condition register, the binary equivalent is 10000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

The returned value can indicate one or more status events occurred.

The following table contains descriptions of the bits:

Bit	Value	Description
B0	status.MEASUREMENT_SUMMARY_BIT status.MSB	Set summary bit indicates that an enabled measurement event has occurred. Bit 0 decimal value: 1
B1	status.SYSTEM_SUMMARY_BIT status.SSB	Set summary bit indicates that an enabled system event has occurred. Bit 1 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 2 decimal value: 4
B3	status.QUESTIONABLE_SUMMARY_BIT status.QSB	Set summary bit indicates that an enabled questionable event has occurred. Bit 3 decimal value: 8
B4	status.MESSAGE_AVAILABLE status.MAV	Set summary bit indicates that a response message is present in the Output Queue. Bit 4 decimal value: 16
B5	status.EVENT_SUMMARY_BIT status.ESB	Set summary bit indicates that an enabled standard event has occurred. Bit 5 decimal value: 32
B6	Not used.	Not applicable.
B7	status.OPERATION_SUMMARY status.OSB	Set summary bit indicates that an enabled operation event has occurred. Bit 7 decimal value: 128

In addition to the above constants, *requestEventRegister* can be set to the decimal equivalent of the bit(s) 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

<pre>requestEventRegister = status.request_event print(requestEventRegister)</pre>	<p>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).</p>
------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

- [status.condition](#) (on page 7-173)
- [status.system.*](#) (on page 7-190)
- [Status byte and service request \(SRQ\)](#) (on page C-15)

status.reset()

This function resets all bits in the system 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.

Example

<pre>status.reset()</pre>	Resets the system status model.
---------------------------	---------------------------------

Also see

None

status.standard.*

These attributes manage the status model's standard 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	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
```

<i>standardRegister</i>	The standard event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.
-------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

These attributes are used to read or write to the standard event status registers. They are set using a constant or a decimal value, but are returned as a decimal 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, if a value of $1.29000e+02$ (which is 129) is read as the value of the condition register, the binary equivalent is 1000001. This value indicates that bit B0 and bit B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
Most significant bit	>	>	>	>	>	>	Least significant bit
1	0	0	0	0	0	0	1

Assigning a value to this attribute enables one or more status events. When an enabled status event occurs, the corresponding bit is set in the enable register.

The individual bits of the status byte condition register can be set using constants (for example, to set the enable bit of the standard event status register to B0, set `status.standard.enable = status.standard.OPC`). The following table contains descriptions of the bits and available values:

Bit	Value	Description
B0	<code>status.standard.OPERATION_COMPLETE</code> <code>status.standard.OPC</code>	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 ICL function <code>opc()</code> can be used in place of the *OPC command. Bit 0 decimal value: 1
B1	Not used	Not applicable
B2	<code>status.standard.QUERY_ERROR</code> <code>status.standard.QYE</code>	Set bit indicates that you attempted to read data from an empty Output Queue. Bit 2 decimal value: 4
B3	<code>status.standard.DEVICE_DEPENDENT_ERROR</code> <code>status.standard.DDE</code>	Set bit indicates that an instrument operation did not execute properly due to some internal condition. Bit 3 decimal value: 8
B4	<code>status.standard.EXECUTION_ERROR</code> <code>status.standard.EXE</code>	Set bit indicates that the instrument detected an error while trying to execute a command. Bit 4 decimal value: 16
B5	<code>status.standard.COMMAND_ERROR</code> <code>status.standard.CME</code>	Set bit indicates that a command error has occurred. Command errors include: IEEE-488.2 syntax error: 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 5 decimal value: 32
B6	<code>status.standard.USER_REQUEST</code> <code>status.standard.URQ</code>	Set bit indicates that the LOCAL key on the instrument front panel was pressed. Bit 6 decimal value: 64
B7	<code>status.standard.POWER_ON</code> <code>status.standard.PON</code>	Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read. Bit 7 decimal value: 128

In addition to the above constants, `standardRegister` can be set to the decimal 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

<pre>standardRegister = status.standard.OPC + status.standard.EXE status.standard.enable = standardRegister</pre>	<p>Sets the OPC and EXE bits of the standard event status register using constants.</p>
-------------------------------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Sets the OPC and EXE bits of the standard event status register using a decimal value.

Also see

[Event summary bit \(ESB register\)](#) (on page C-9)

status.system.*

These attributes manage the status model's TSP-Link system summary register for nodes 1 through 14.

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	32767 (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
```

<i>enableRegister</i>	The system event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.
-----------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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 decimal value, but are returned as a decimal 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

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. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B0 of the system summary status register, set `status.system.enable = status.system.enable.EXT`):

Bit	Value	Description
B0	<code>status.system.EXTENSION_BIT</code> <code>status.system.EXT</code>	Bit 0 decimal value: 1
B1	<code>status.system.NODE1</code>	Bit 1 decimal value: 2
B2	<code>status.system.NODE2</code>	Bit 2 decimal value: 4
B3	<code>status.system.NODE3</code>	Bit 3 decimal value: 8
B4	<code>status.system.NODE4</code>	Bit 4 decimal value: 16
B5	<code>status.system.NODE5</code>	Bit 5 decimal value: 32
B6	<code>status.system.NODE6</code>	Bit 6 decimal value: 64
B7	<code>status.system.NODE7</code>	Bit 7 decimal value: 128
B8	<code>status.system.NODE8</code>	Bit 8 decimal value: 256
B9	<code>status.system.NODE9</code>	Bit 9 decimal value: 512
B10	<code>status.system.NODE10</code>	Bit 10 decimal value: 1024
B11	<code>status.system.NODE11</code>	Bit 11 decimal value: 2048
B12	<code>status.system.NODE12</code>	Bit 12 decimal value: 4096
B13	<code>status.system.NODE13</code>	Bit 13 decimal value: 8192
B14	<code>status.system.NODE14</code>	Bit 14 decimal value: 16384
B15	Not used	Not applicable

In addition to the above constants, `enableRegister` can be set to the decimal 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 2048 + 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	1024	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
```

Sets BIT11 and BIT14 of the status system enable register set 1 using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system.enable = enableRegister
```

Sets BIT11 and BIT14 of the status system enable register set 1 register using a decimal value.

Also see

[status.system2.*](#) (on page 7-192)

[System summary bit \(System register\)](#) (on page C-5)

status.system2.*

These attributes manage the status model's TSP-Link system summary register for nodes 15 through 28.

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	32767 (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 system event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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 decimal value, but are returned as a decimal 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

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. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B0 of the system summary status register set 2, set `status.system2.enable = status.system2.EXT`):

Bit	Value	Description
B0	<code>status.system2.EXTENSION_BIT</code> <code>status.system2.EXT</code>	Bit 0 decimal value: 1
B1	<code>status.system2.NODE15</code>	Bit 1 decimal value: 2
B2	<code>status.system2.NODE16</code>	Bit 2 decimal value: 4
B3	<code>status.system2.NODE17</code>	Bit 3 decimal value: 8
B4	<code>status.system2.NODE18</code>	Bit 4 decimal value: 16
B5	<code>status.system2.NODE19</code>	Bit 5 decimal value: 32
B6	<code>status.system2.NODE20</code>	Bit 6 decimal value: 64
B7	<code>status.system2.NODE21</code>	Bit 7 decimal value: 128
B8	<code>status.system2.NODE22</code>	Bit 8 decimal value: 256
B9	<code>status.system2.NODE23</code>	Bit 9 decimal value: 512
B10	<code>status.system2.NODE24</code>	Bit 10 decimal value: 1024
B11	<code>status.system2.NODE25</code>	Bit 11 decimal value: 2048
B12	<code>status.system2.NODE26</code>	Bit 12 decimal value: 4096
B13	<code>status.system2.NODE27</code>	Bit 13 decimal value: 8192
B14	<code>status.system2.NODE28</code>	Bit 14 decimal value: 16384
B15	Not used	Not applicable

In addition to the above constants, *enableRegister* can be set to the decimal 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 2048 + 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	1024	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
```

Sets BIT11 and BIT14 of the status system enable register set 2 using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system2.enable = enableRegister
```

Sets BIT11 and BIT14 of the status system enable register set 2 register using a decimal value.

Also see

[status.system.*](#) (on page 7-190)

[status.system3.*](#) (on page 7-194)

[System summary bit \(System register\)](#) (on page C-5)

status.system3.*

These attributes manage the status model's TSP-Link system summary register for nodes 29 through 42.

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	32767 (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 system event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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 decimal value, but are returned as a decimal 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 10000001. 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

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. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B0 of the system summary status register set 3, set `status.system3.enable = status.system3.EXT`):

Bit	Value	Description
B0	<code>status.system3.EXTENSION_BIT</code> <code>status.system3.EXT</code>	Bit 0 decimal value: 1
B1	<code>status.system3.NODE29</code>	Bit 1 decimal value: 2
B2	<code>status.system3.NODE30</code>	Bit 2 decimal value: 4
B3	<code>status.system3.NODE31</code>	Bit 3 decimal value: 8
B4	<code>status.system3.NODE32</code>	Bit 4 decimal value: 16
B5	<code>status.system3.NODE33</code>	Bit 5 decimal value: 32
B6	<code>status.system3.NODE34</code>	Bit 6 decimal value: 64
B7	<code>status.system3.NODE35</code>	Bit 7 decimal value: 128
B8	<code>status.system3.NODE36</code>	Bit 8 decimal value: 256
B9	<code>status.system3.NODE37</code>	Bit 9 decimal value: 512
B10	<code>status.system3.NODE38</code>	Bit 10 decimal value: 1024
B11	<code>status.system3.NODE39</code>	Bit 11 decimal value: 2048
B12	<code>status.system3.NODE40</code>	Bit 12 decimal value: 4096
B13	<code>status.system3.NODE41</code>	Bit 13 decimal value: 8192
B14	<code>status.system3.NODE42</code>	Bit 14 decimal value: 16384
B15	Not used	Not applicable

In addition to the above constants, *enableRegister* can be set to the decimal 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 2048 + 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	1024	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
```

Sets BIT11 and BIT14 of the status system enable register set 3 using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system3.enable = enableRegister
```

Sets BIT11 and BIT14 of the status system enable register set 3 register using a decimal value.

Also see

[status.system2.*](#) (on page 7-192)

[status.system4.*](#) (on page 7-196)

[System summary bit \(System register\)](#) (on page C-5)

status.system4.*

These attributes manage the status model's TSP-Link system summary register for nodes 43 through 56.

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	32767 (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 system event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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 decimal value, but are returned as a decimal 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

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. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B0 of the system summary status register set 4, set `status.system4.enable = status.system4.enable.EXT`):

Bit	Value	Description
B0	<code>status.system4.EXTENSION_BIT</code> <code>status.system4.EXT</code>	Bit 0 decimal value: 1
B1	<code>status.system4.NODE43</code>	Bit 1 decimal value: 2
B2	<code>status.system4.NODE44</code>	Bit 2 decimal value: 4
B3	<code>status.system4.NODE45</code>	Bit 3 decimal value: 8
B4	<code>status.system4.NODE46</code>	Bit 4 decimal value: 16
B5	<code>status.system4.NODE47</code>	Bit 5 decimal value: 32
B6	<code>status.system4.NODE48</code>	Bit 6 decimal value: 64
B7	<code>status.system4.NODE49</code>	Bit 7 decimal value: 128
B8	<code>status.system4.NODE50</code>	Bit 8 decimal value: 256
B9	<code>status.system4.NODE51</code>	Bit 9 decimal value: 512
B10	<code>status.system4.NODE52</code>	Bit 10 decimal value: 1024
B11	<code>status.system4.NODE53</code>	Bit 11 decimal value: 2048
B12	<code>status.system4.NODE54</code>	Bit 12 decimal value: 4096
B13	<code>status.system4.NODE55</code>	Bit 13 decimal value: 8192
B14	<code>status.system4.NODE56</code>	Bit 14 decimal value: 16384
B15	Not used	Not applicable

In addition to the above constants, *enableRegister* can be set to the decimal 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 2048 + 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	1024	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
```

Sets BIT11 and BIT14 of the status system enable register set 4 using constants.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system4.enable = enableRegister
```

Sets BIT11 and BIT14 of the status system enable register set 4 register using a decimal value.

Also see

[status.system3.*](#) (on page 7-194)

[status.system5.*](#) (on page 7-198)

[System summary bit \(System register\)](#) (on page C-5)

status.system5.*

These attributes manage the status model's TSP-Link system summary register for nodes 57 through 64.

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

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 system event register's status in decimal form. A zero (0) indicates no bits set (also send 0 to clear all bits). Other decimal values indicate various bit settings.

Details

Although the status logical instrument reset affects these attributes, a system reset does not.

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 decimal value, but are returned as a decimal 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 0001. 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

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. The individual bits of the status byte condition register can be set to the following described values (for example, to set the enable bit to B1 of the system summary status register set 5, set `status.system5.enable = status.system5.NODE57`):

Bit	Value	Description
B0	Not used	Not applicable
B1	<code>status.system5.NODE57</code>	Bit 1 decimal value: 2
B2	<code>status.system5.NODE58</code>	Bit 2 decimal value: 4
B3	<code>status.system5.NODE59</code>	Bit 3 decimal value: 8
B4	<code>status.system5.NODE60</code>	Bit 4 decimal value: 16
B5	<code>status.system5.NODE61</code>	Bit 5 decimal value: 32
B6	<code>status.system5.NODE62</code>	Bit 6 decimal value: 64
B7	<code>status.system5.NODE63</code>	Bit 7 decimal value: 128
B8	<code>status.system5.NODE64</code>	Bit 8 decimal value: 256
B9-B14	Not used	Not applicable

In addition to the above constants, `enableRegister` can be set to the decimal 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	1024	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
```

Sets BIT1 and BIT5 of the status system enable register set 5 using constants.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Sets BIT1 and BIT5 of the status system enable register set 5 register using a decimal value.

Also see

[status.system4.*](#) (on page 7-196)

[System summary bit \(System register\)](#) (on page C-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

Returns the elapsed time in seconds (1 μ s resolution)

Details

The returned resolution for time depends on how long it has been since the timer was reset. It starts with 1 μ s resolution and starts to lose resolution after about 2.8 minutes.

Example 1

```
timer.reset()
...
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.beeper(0.5, 2400)
print("reset timer")
timer.reset()
delay(0.5)
dt = timer.measure.t()
print("timer after delay:", dt)
beeper.beeper(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 7-201)

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()
...
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 7-200)

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 function sets the blender event detector to the undetected state and resets the event detector's overrun indicator.

Also see

None

trigger.blender[N].EVENT_ID

This attribute sets the trigger blender event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.blender[N].EVENT_ID
```

<i>eventID</i>	Trigger event number
<i>N</i>	The blender number (1 or 2)

Details

Set the stimulus of any trigger event detector to the value of this constant to have it respond to trigger events from this trigger blender.

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	System reset Trigger blender N reset	Create configuration script	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

<i>orenable</i>	The orenable mode: <ul style="list-style-type: none"> • true: OR mode • false: AND mode
<i>N</i>	The trigger blender (1 or 2)

Details

This attribute 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.

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 7-203)

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	System reset Trigger blender N reset	Not applicable	Not applicable

Usage

```
overrun = trigger.blender[N].overrun
```

<i>overrun</i>	Trigger blender overrun state
<i>N</i>	The trigger event blender (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 attribute 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 the event was not ignored, the output is `false`.

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 7-203)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

<i>N</i>	The trigger event blender (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`.

Also see

[trigger.blender\[N\].orenable](#) (on page 7-202)

[trigger.blender\[N\].overrun](#) (on page 7-203)

[trigger.blender\[N\].stimulus\[M\]](#) (on page 7-204)

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	System reset Trigger blender N reset	Create configuration script	0

Usage

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

<i>eventID</i>	The event that triggers the blender action; see Details
<i>N</i>	The trigger event blender (1 or 2)
<i>M</i>	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
digio.trigger[N].EVENT_ID	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
display.trigger.EVENT_ID	The trigger key on the front panel is pressed
trigger.EVENT_ID	A *trg message on the active command interface. If GPIB is the active command interface, a GET message 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_SCAN_COMP	Scan complete event
scan.trigger.EVENT_IDLE	Idle event

Use zero to disable the blender input.

NOTE	
Use the <i>eventID</i> parameter rather than the numeric value to set the stimulus value. This will help make the code compatible for future upgrades.	

Example

<pre> 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 scan.trigger.arm.stimulus = trigger.blender[1].EVENT_ID </pre>	<p>Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.</p> <p>Set the trigger blender 1 event to start a scan.</p>
--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[trigger.blender\[N\].reset\(\)](#) (on page 7-203)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.blender[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication for blender
<i>N</i>	The trigger blender (1 or 2) on which to wait
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```

digio.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

print(trigger.blender[1].wait(3))

```

Generate a trigger blender 1 event when a digital I/O trigger happens either on line 3 or 5.

Wait three seconds while checking if trigger blender 1 event has occurred.

If the blender trigger event has happened, then `true` is returned. If the trigger event has not happened, then `false` is returned after the timeout expires.

Also see

[trigger.blender\[N\].clear\(\)](#) (on page 7-201)

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

Also see

[trigger.wait\(\)](#) (on page 7-214)

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
```

<code>eventID</code>	The command interface trigger event number
----------------------	--------------------------------------------

Details

You can set the stimulus of any trigger event detector to the value of this constant to have it respond to command interface trigger events.

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()
```

<i>N</i>	Trigger timer number to clear (1 to 4)
----------	----------------------------------------

Details

This function sets the timer event detector to the undetected state and resets the overrun indicator.

Also see

[trigger.timer\[N\].count](#) (on page 7-207)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer is triggered.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Trigger timer N reset	Create configuration script	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

<i>count</i>	Number of times to repeat the trigger
<i>N</i>	A trigger timer value from 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 retrigger count for timer number 1.

Also see

[trigger.timer\[N\].clear\(\)](#) (on page 7-207)

[trigger.timer\[N\].reset\(\)](#) (on page 7-211)

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	System reset Trigger timer N reset	Create configuration script	10e-6

Usage

```
interval = trigger.timer[N].delay
```

```
trigger.timer[N].delay = interval
```

<i>interval</i>	Delay interval in seconds
<i>N</i>	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 μ s.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 7-211)

trigger.timer[N].delaylist

This attribute sets an array of timer intervals that are used when triggered.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Trigger timer N reset	Create configuration script	{10e-6}

Usage

```
intervals = trigger.timer[N].delaylist
trigger.timer[N].delaylist = intervals
```

<i>intervals</i>	Table of delay intervals in seconds
<i>N</i>	Trigger timer number (1 to 4)

Details

Each time the timer is triggered, it uses the next delay period from the array.

After all elements in the array have been used, the delays restart at the beginning of the list.

Example

<pre>trigger.timer[3].delaylist = {50e-6, 100e-6, 150e-6} DelayList = trigger.timer[3].delaylist for x = 1, table.getn(DelayList) do print(DelayList[x]) end</pre>	<p>Set a delay list on trigger timer 3 with three delays (50 μs, 100 μs, and 150 μs).</p> <p>Read the delay list on trigger timer 3:</p> <p>Output (assuming the list was set with three delays (50 μs, 100-μs, and 150 μs):</p> <pre>5.000000000e-05 1.000000000e-04 1.500000000e-04</pre>
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 7-211)

trigger.timer[N].EVENT_ID

This constant contains the trigger timer event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.timer[N].EVENT_ID
```

<i>eventID</i>	The trigger event number
<i>N</i>	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 event detector to the value of this constant to have it respond to events from this timer.

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	System reset Trigger timer N reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

<i>overrun</i>	Trigger overrun state
<i>N</i>	Trigger timer value (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 7-211)

trigger.timer[N].passthrough

This attribute enables or disables the timer trigger passthrough.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset Trigger timer N reset	Create configuration script	False

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

<i>passthrough</i>	Passthrough: <ul style="list-style-type: none"> • True: Enabled • False: Disabled
<i>N</i>	Trigger timer number (1 to 4)

Details

When enabled, triggers are passed through immediately and initiate the delay. When disabled, a trigger only initiates a delay.

Also see

[trigger.timer\[N\].reset\(\)](#) (on page 7-211)

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()
```

<i>N</i>	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`.

Also see

- [trigger.timer\[N\].count](#) (on page 7-207)
- [trigger.timer\[N\].delay](#) (on page 7-208)
- [trigger.timer\[N\].delaylist](#) (on page 7-209)
- [trigger.timer\[N\].overrun](#) (on page 7-210)
- [trigger.timer\[N\].passthrough](#) (on page 7-211)
- [trigger.timer\[N\].stimulus](#) (on page 7-212)

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	System reset Trigger timer N reset	Create configuration script	0

Usage

```
eventID = trigger.timer[N].stimulus
trigger.timer[N].stimulus = eventID
```

<i>eventID</i>	The event that triggers the timer delay
<i>N</i>	A number representing a 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
digio.trigger[N].EVENT_ID	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
display.trigger.EVENT_ID	The trigger key on the front panel is pressed
trigger.EVENT_ID	A *trg message on the active command interface. If GPIB is the active command interface, a GET message 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_SCAN_COMP	Scan complete event
scan.trigger.EVENT_IDLE	Idle event

Set this attribute to the *eventID* of any trigger event to wait for that event.
Use zero to disable event processing.

NOTE

Use trigger event ID rather than the numeric value to set the stimulus value. This will help make the code compatible with future upgrades.

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 7-211)

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = trigger.timer[N].wait(timeout)
```

<i>triggered</i>	Trigger detection indication
<i>N</i>	Trigger timer number (1 to 4)
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.timer[3].wait(10)
print(triggered)
```

Waits up to 10 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 7-207)

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)
```

<i>triggered</i>	True: A trigger was detected during the timeout period False: No triggers were detected during the timeout period
<i>timeout</i>	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A GPIB GET command is detected (GPIB only)
- A VXI-11 device_trigger method is invoked (VXI-11 only)
- A *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 7-206)

tsplink.group

This attribute is the group number of a TSP-Link node used for DTNS.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Not saved	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

<i>groupNumber</i>	The group number of the TSP-Link node (0 to 64)
--------------------	-------------------------------------------------

Details

To remove the node from all groups, set the attribute value to 0.
When the node is turned off, the group number for that node changes to 0.

Also see

None

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
masterNodeNumber = tsplink.master
```

<i>masterNodeNumber</i>	The node number of the master node
-------------------------	------------------------------------

Also see

None

tsplink.node

This attribute defines the node number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	2

Usage

```
nodeNumber = tsplink.node  
tsplink.node = nodeNumber
```

<i>nodeNumber</i>	Set node to a number (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 the next time `tsplink.reset()` is executed on any node in the system.
Each node connected to the TSP-Link must be assigned a different node number.

Example

<code>tsplink.node = 2</code>	Sets the TSP-Link node to number 2.
-------------------------------	-------------------------------------

Also see[tsplink.reset\(\)](#) (on page 7-217)[tsplink.state](#) (on page 7-218)

tsplink.readbit()

This function reads the state of a TSP-Link synchronization line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readbit(N)
```

<i>data</i>	A custom variable that stores the state of the synchronization line
<i>N</i>	The trigger line (1 to 3)

Details

Returns a value of 0 if the line is low and 1 if the line is high.

Example

<pre>data = tsplink.readbit(3) print(data)</pre>	Assume Line 3 is set high, and it is then read. Output 1.000000e+00
--------------------------------------------------	---------------------------------------------------------------------------

Also see[tsplink.readport\(\)](#) (on page 7-216)[tsplink.writebit\(\)](#) (on page 7-226)

tsplink.readport()

This function reads the TSP-Link synchronization lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
```

<i>data</i>	Numeric value returned indicating which register bits 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 Bit 3 corresponds to Line 3. For example, a returned value of 2 has a binary equivalent of 010. Line 2 is high (1), and the other 2 lines are low (0).

Example

```
data = tsplink.readport()
print(data)
```

Reads state of all three TSP lines.
Assuming Line 2 is set high, the output is:
2.000000e+00
(binary 010)

Also see

[tsplink.readbit\(\)](#) (on page 7-216)

[tsplink.writebit\(\)](#) (on page 7-226)

[tsplink.writeport\(\)](#) (on page 7-226)

tsplink.reset()

This function initializes (resets) all nodes (instruments) in the TSP-Link system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
nodesFound = tsplink.reset()
nodesFound = tsplink.reset(expectedNodes)
```

<i>nodesFound</i>	The number of nodes actually found on the system
<i>expectedNodes</i>	The number of nodes expected on the system, between 1 and 64

Details

This function erases all knowledge of other nodes connected on the TSP-Link 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 initialized 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.

Also see

[tsplink.node](#) (on page 7-215)

[tsplink.state](#) (on page 7-218)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
state = tsplink.state
```

<code>state</code>	TSP-Link state, online or offline
--------------------	-----------------------------------

Details

When the instrument is powered on, the state is `offline`. After `tsplink.reset()` is successful, the state is `online`.

Example

<pre>state = tsplink.state print(state)</pre>	Read the state of the TSP-Link. If it is online, the output is: <code>online</code>
-----------------------------------------------	----------------------------------------------------------------------------------------

Also see

[tsplink.node](#) (on page 7-215)
[tsplink.reset\(\)](#) (on page 7-217)

tsplink.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

```
tsplink.trigger[N].assert()
```

<code>N</code>	The trigger line (1 to 3)
----------------	---------------------------

Details

The set pulse width determines how long the trigger is asserted.

Example

<pre>tsplink.trigger[2].assert()</pre>	Asserts trigger on trigger line 2.
----------------------------------------	------------------------------------

Also see

[tsplink.trigger\[N\].clear\(\)](#) (on page 7-219)
[tsplink.trigger\[N\].mode](#) (on page 7-220)
[tsplink.trigger\[N\].overrun](#) (on page 7-221)

[tsplink.trigger\[N\].pulsewidth](#) (on page 7-222)

[tsplink.trigger\[N\].release\(\)](#) (on page 7-223)

[tsplink.trigger\[N\].stimulus](#) (on page 7-224)

[tsplink.trigger\[N\].wait\(\)](#) (on page 7-225)

tsplink.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

```
tsplink.trigger[N].clear()
```

<i>N</i>	The trigger line (1 to 3)
----------	---------------------------

Details

The event detector for a trigger recalls if a trigger event has been detected since the last `tsplink.trigger[N].wait()` call. This function clears a trigger event detector, discards the previous history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.

Example

<code>tsplink.trigger[2].clear()</code>	Clears trigger event on synchronization Line 2.
-----------------------------------------	-------------------------------------------------

Also see

[tsplink.trigger\[N\].mode](#) (on page 7-220)

[tsplink.trigger\[N\].overrun](#) (on page 7-221)

[tsplink.trigger\[N\].release\(\)](#) (on page 7-223)

[tsplink.trigger\[N\].stimulus](#) (on page 7-224)

[tsplink.trigger\[N\].wait\(\)](#) (on page 7-225)

tsplink.trigger[N].EVENT_ID

This attribute identifies the number that is used for the trigger events.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = tsplink.trigger[N].EVENT_ID
```

<i>eventID</i>	The trigger event number.
<i>N</i>	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 event detector to the value of this constant to have it respond to trigger events from this line.

Also see

None

tsplink.trigger[N].mode

This attribute defines the trigger operation and detection mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset TSP-Link trigger N reset	Create configuration script	TRIG_BYPASS

Usage

```
mode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = mode
```

<i>mode</i>	The trigger mode
<i>N</i>	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 values		
Mode	Number value	Description
<code>tsplink.TRIG_BYPASS</code>	0	Allows direct control of the line as a digital I/O line.
<code>tsplink.TRIG_FALLING</code>	1	Detects falling-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISING</code>	2	If the programmed state of the line is high, the <code>tsplink.TRIG_RISING</code> mode behaves similar to <code>tsplink.TRIG_RISINGA</code> . If the programmed state of the line is low, the <code>tsplink.TRIG_RISING</code> mode behaves similar to <code>tsplink.TRIG_RISINGM</code> . Use <code>digio.TRIG_RISINGA</code> if the line is in the high output state. Use <code>digio.TRIG_RISINGM</code> if the line is in the low output state.
<code>tsplink.TRIG_EITHER</code>	3	Detects rising- or falling-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_SYNCHRONOUSA</code>	4	Detects the falling-edge input triggers and automatically latches and drives the trigger line low.
<code>tsplink.TRIG_SYNCHRONOUS</code>	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.

<code>tsplink.TRIG_SYNCHRONOUSM</code>	6	Detects rising-edge triggers as an input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISINGA</code>	7	Detects rising-edge triggers as input. Asserts a TTL-low pulse for output.
<code>tsplink.TRIG_RISINGM</code>	8	Edge detection as an input is not available. Generates a TTL-high pulse as an output trigger.

When programmed to any other mode, the output state of the I/O line is controlled by the trigger logic, and the user-specified output state of the line is ignored.

When the trigger mode is set to `tsplink.TRIG_RISING`, the user-specified output state of the line will be examined. If the output state selected when the mode is changed is high, the actual mode used will be `tsplink.TRIG_RISINGA`. If the output state selected when the mode is changed is low, the actual mode used will be `tsplink.TRIG_RISINGM`.

The custom variable 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 the synchronization Line 3 to `tsplink.TRIG_RISINGM`.

Also see

- [digio.writebit\(\)](#) (on page 7-64)
- [digio.writeport\(\)](#) (on page 7-65)
- [tsplink.trigger\[N\].assert\(\)](#) (on page 7-218)
- [tsplink.trigger\[N\].clear\(\)](#) (on page 7-219)
- [tsplink.trigger\[N\].overrun](#) (on page 7-221)
- [tsplink.trigger\[N\].release\(\)](#) (on page 7-223)
- [tsplink.trigger\[N\].reset\(\)](#) (on page 7-223)
- [tsplink.trigger\[N\].stimulus](#) (on page 7-224)
- [tsplink.trigger\[N\].wait\(\)](#) (on page 7-225)

tsplink.trigger[N].overrun

This attribute indicates if the event detector ignored an event while in the detected state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	System reset TSP-Link trigger N reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

<code>overrun</code>	Trigger overrun state
<code>N</code>	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 7-218)

[tsplink.trigger\[N\].clear\(\)](#) (on page 7-219)

[tsplink.trigger\[N\].mode](#) (on page 7-220)

[tsplink.trigger\[N\].release\(\)](#) (on page 7-223)

[tsplink.trigger\[N\].reset\(\)](#) (on page 7-223)

[tsplink.trigger\[N\].stimulus](#) (on page 7-224)

[tsplink.trigger\[N\].wait\(\)](#) (on page 7-225)

tsplink.trigger[N].pulsewidth

This attribute the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset TSP-Link trigger N reset	Create configuration script	10-e6

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

<i>width</i>	The pulse width (in seconds).
<i>N</i>	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 μ s.

Also see

[tsplink.trigger\[N\].release\(\)](#) (on page 7-223)

tsplink.trigger[N].release()

This function releases a latched trigger on the given TSP-Link trigger line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].release()
```

<i>N</i>	The trigger line (1 to 3)
----------	---------------------------

Details

Releases a trigger that was asserted with an indefinite pulse width, as well as a trigger that was latched in response to receiving a synchronous mode trigger.

Example

<code>tsplink.trigger[3].release()</code>	Releases trigger Line 3.
-------------------------------------------	--------------------------

Also see

- [tsplink.trigger\[N\].assert\(\)](#) (on page 7-218)
- [tsplink.trigger\[N\].clear\(\)](#) (on page 7-219)
- [tsplink.trigger\[N\].mode](#) (on page 7-220)
- [tsplink.trigger\[N\].overrun](#) (on page 7-221)
- [tsplink.trigger\[N\].pulsewidth](#) (on page 7-222)
- [tsplink.trigger\[N\].stimulus](#) (on page 7-224)
- [tsplink.trigger\[N\].wait\(\)](#) (on page 7-225)

tsplink.trigger[N].reset()

This function resets some of the TSP-Link trigger settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].reset()
```

<i>N</i>	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`.

Also see

[tsplink.trigger\[N\].pulsewidth](#) (on page 7-222)

[tsplink.trigger\[N\].mode](#) (on page 7-220)

[tsplink.trigger\[N\].overrun](#) (on page 7-221)

[tsplink.trigger\[N\].stimulus](#) (on page 7-224)

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	System reset TSP-Link trigger N reset	Create configuration script	0

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

<i>eventID</i>	The event identifier for the triggering event
<i>N</i>	The trigger line (1 to 3)

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to 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
<code>digio.trigger[N].EVENT_ID</code>	An edge (either rising, falling, or either based on the configuration of the line) on the digital input line
<code>display.trigger.EVENT_ID</code>	The trigger key on the front panel is pressed
<code>trigger.EVENT_ID</code>	A *trg message on the active command interface. If GPIB is the active command interface, a GET message also generates this event
<code>trigger.blender[N].EVENT_ID</code>	A combination of events has occurred
<code>trigger.timer[N].EVENT_ID</code>	A delay expired
<code>tsplink.trigger[N].EVENT_ID</code>	An edge (either rising, falling, or either based on the configuration of the line) on the TSP-Link trigger line
<code>lan.trigger[N].EVENT_ID</code>	A LAN trigger event has occurred
<code>scan.trigger.EVENT_SCAN_READY</code>	Scan ready event
<code>scan.trigger.EVENT_SCAN_START</code>	Scan start event
<code>scan.trigger.EVENT_CHANNEL_READY</code>	Channel ready event
<code>scan.trigger.EVENT_SCAN_COMP</code>	Scan complete event
<code>scan.trigger.EVENT_IDLE</code>	Idle event

NOTE

Use the constant for `EVENT_ID` rather than the numeric value to set the stimulus. This will help make the code compatible for future upgrades.

Also see

[tsplink.trigger\[N\].assert\(\)](#) (on page 7-218)

[tsplink.trigger\[N\].reset\(\)](#) (on page 7-223)

tsplink.trigger[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
triggered = tsplink.trigger[N].wait(timeout)
```

<code>triggered</code>	Trigger detection indication: <ul style="list-style-type: none"> • <code>True</code>: A trigger is detected during the timeout period • <code>False</code>: A trigger is not detected during the timeout period
<code>N</code>	The trigger line (1 to 3)
<code>timeout</code>	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 7-219)

tsplink.writebit()

This function sets a TSP-Link synchronization line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writebit(bit, data)
```

<i>bit</i>	The trigger line (1 to 3)
<i>data</i>	The value to write to the bit: <ul style="list-style-type: none"> • Low: 0 • High: 1

Details

Use `tsplink.writebit()` and `tsplink.writeport()` to control the output state of the synchronization 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 digital I/O lines.

Example

```
tsplink.writebit(3, 0)    Sets synchronization Line 3 low (0).
```

Also see

[tsplink.readbit\(\)](#) (on page 7-216)

[tsplink.readport\(\)](#) (on page 7-216)

[tsplink.writeport\(\)](#) (on page 7-226)

tsplink.writeport()

This function writes to all TSP-Link synchronization lines.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.writeport(data)
```

<i>data</i>	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 (see `tsplink.writeprotect` (on page 7-227)).

The `reset()` function does not affect the present states of the digital I/O lines.

Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when 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 7-216)
[tsplink.readport\(\)](#) (on page 7-216)
[tsplink.writebit\(\)](#) (on page 7-226)
[tsplink.writeprotect](#) (on page 7-227)

tsplink.writeprotect

This attribute contains the write-protect mask that protects bits from changes by the `tsplink.writebit()` and `tsplink.writeport()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset	Create configuration script	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

<i>mask</i>	Specifies the value of the bit pattern for write-protect; bits set to 1 to write-protect the corresponding line
-------------	-----------------------------------------------------------------------------------------------------------------

Details

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a *mask* value of 5 has a binary equivalent 101. This *mask* write-protects Lines 1 and 3.

Example

```
tsplink.writeprotect = 5           Write-protects Lines 1 and 3.
```

Also see

[tsplink.readbit\(\)](#) (on page 7-216)
[tsplink.readport\(\)](#) (on page 7-216)
[tsplink.writeport\(\)](#) (on page 7-226)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.clear(connectionID)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
---------------------	---------------------------------------------------------------

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

<pre>tspnet.write(mydevice, "print([[hello]])") print(tspnet.readavailable(mydevice))</pre>	<p>Write data to device, print how much is available.</p> <p>Output: 6.0000000e+000</p>
<pre>tspnet.clear(mydevice) print(tspnet.readavailable(mydevice))</pre>	<p>Clear data and print how much data is available again.</p> <p>Output: 0.0000000e+000</p>

Also see

[tspnet.connect\(\)](#) (on page 7-229)
[tspnet.readavailable\(\)](#) (on page 7-233)
[tspnet.write\(\)](#) (on page 7-240)

tspnet.connect()

This function connects the device processing the command to another device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
connectionID = tspnet.connect(ipAddress)
connectionID = tspnet.connect(ipAddress, portNumber)
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

<i>connectionID</i>	The connection ID to be used as a handle in all other <code>tspnet</code> function calls
<i>ipAddress</i>	IP address to which to connect
<i>portNumber</i>	Port number
<i>initString</i>	Initialization string to send to <i>ipAddress</i>

Details

This command connects a device to another device through the LAN interface. The default port number is 5025. 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 Model 707B or 708B does not perform any extra processing, prompt handling, error handling, or sending of commands. Additionally, 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 `tspnet.tsp.abortonconnect`, the Model 707B or 708B sends an `abort` command to the remote device on connection. The Model 707B or 708B also enables TSP prompts on the remote device and error management. The Model 707B or 708B 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, or subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
myID = tspnet.connect("192.0.2.1")
if myID then
  -- Use myID as needed here
end
tspnet.disconnect(myID)
```

Connect to a TSP-enabled device.

Example 2

```
myID = tspnet.connect("192.0.2.1", 1394,
  "*rst\r\n")
if myID then
  -- Use myID as needed here
end
tspnet.disconnect(myID)
```

Connect to a device that is not TSP-enabled.

Also see

[localnode.prompts](#) (on page 7-129)
[localnode.showerrors](#) (on page 7-133)
[tspnet.tsp.abortonconnect](#) (on page 7-237)
[tspnet.disconnect\(\)](#) (on page 7-230)

tspnet.disconnect()

This function disconnects the TSP-Net session specified.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.disconnect(connectionID)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
---------------------	---------------------------------------------------------------

Details

This function disconnects the two devices by closing the connection. The *connectionID* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

<pre>testID = tspnet.connect("192.0.2.0") -- Use the connection tspnet.disconnect(testID)</pre>	<p>Create a TSP-Net session.</p> <p>Close the session.</p>
--------------------------------------------------------------------------------------------------	------------------------------------------------------------

Also see

[tspnet.connect\(\)](#) (on page 7-229)

tspnet.execute()

This function executes a command string on the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```

results = tspnet.execute(connectionID, commandString)
results = tspnet.execute(connectionID, commandString, formatString)
value1 = tspnet.execute(connectionID, commandString, formatString)
value1, value2 = tspnet.execute(connectionID, commandString, formatString)
value1, ..., valuen = tspnet.execute(connectionID, commandString, formatString)

```

<i>results</i>	The results of the command execution
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>commandString</i>	The command to send to the remote device
<i>formatString</i>	Format string for the output
<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valuen</i>	The nth value decoded from the response message; there is one return value per format specifier in the format string

Details

This command sends *commandString* to the remote device connection represented by *connectionID*. The configured termination sequence is added to *commandString* when it is sent to the device (`tspnet.termination()`). When *formatString* is specified, the command waits for a return string from the device. The Model 707B or 708B decodes the output string according to the format specified in *formatString* and returns this output string as arguments from the function.

When this command is sent to a TSP-enabled device, the Model 707B or 708B blocks operation until the device responds or until a timeout error is generated. The TSP prompt from the remote device is read and thrown away. The Model 707B or 708B places any remotely generated errors into its error queue. When the optional *formatString* is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the line.

Example 1

```
tspnet.execute(myID, "runScript()")
```

Command remote device to run script named runScript

Example 2

```

tspnet.termination(myID, tspnet.TERM_CRLF)
tspnet.execute(myID, "*idn?")
print("tspnet.execute returns:",
      tspnet.read(myID))

```

Print the *idn? string from the remote device.

Also see

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.read\(\)](#) (on page 7-232)

[tspnet.write\(\)](#) (on page 7-240)

tspnet.idn()

This function retrieves the response of the remote device to *IDN.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
idnString = tspnet.idn(connectionID)
```

<i>idnString</i>	The returned *IDN string
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This function retrieves the response of the remote device to *IDN.

Example

<pre>myID = tspnet.connect("192.0.2.1") print(tspnet.idn(myID)) tspnet.disconnect(myID)</pre>	<p>Output:</p> <pre>KEITHLEY INSTRUMENTS INC.,MODEL 707B,00000170,01.10h</pre>
-------------------------------------------------------------------------------------------------	--------------------------------------------------------------------------------

Also see

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.disconnect\(\)](#) (on page 7-230)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
results = tspnet.read(connectionID)
results = tspnet.read(connectionID, formatString)
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valuen = tspnet.read(connectionID, formatString)
```

<i>results</i>	The return code from the function call
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>formatString</i>	Format string for the output, maximum of 10 specifiers
<i>value1</i>	The first value decoded from the response message
<i>value2</i>	The second value decoded from the response message
<i>valuen</i>	The nth value decoded from the response message; there is one return value per format specifier in the format string

Details

This command reads available data from the device and returns the number of arguments.

The format string can contain the following identifiers:

<code>%[width]s</code>	Read data until the specific length
<code>%[max width]t</code>	Read data until the specific length or delimited by punctuation
<code>%[max width]n</code>	Read data until a newline or carriage return
<code>%d</code>	Read a number (delimited by punctuation)

If *formatString* is not provided, the command returns a string containing the data until a new line is reached. If no data is available, the Model 707B or 708B holds off operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When reading from a TSP-enabled remote device, the Model 707B or 708B removes TSP prompts and places any errors received from the remote device into its own error queue. The Model 707B or 708B prefaces errors from the remote device with "Remote Error," and follows this with the error number and error description.

Example

<pre>tspnet.write(myID, "*idn?\r\n") print("write/read returns:", tspnet.read(myID))</pre>	<p>Send the "<code>*idn?\r\n</code>" message to the device connected as <code>myID</code>.</p> <p>Display the response that is read from <code>myID</code> (based on the <code>*idn?</code> message).</p>
--------------------------------------------------------------------------------------------	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Also see

[tspnet.connect\(\)](#) (on page 7-229)
[tspnet.disconnect\(\)](#) (on page 7-230)
[tspnet.timeout](#) (on page 7-236)
[tspnet.write\(\)](#) (on page 7-240)

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	Yes			

Usage

```
bytesAvailable = tspnet.readavailable(connectionID)
```

<i>bytesAvailable</i>	The return code from the function call
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>

Details

This command checks to see if any output data is available from the device. No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```

ID = tspnet.connect("192.0.2.1")

tspnet.write(ID, "*idn?\r\n")

repeat

    bytes = tspnet.readavailable(ID)
until bytes > 0

print(tspnet.read(ID))
tspnet.disconnect(ID)

```

Send commands that will create data.

Wait for data to be available.

Also see

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.disconnect\(\)](#) (on page 7-230)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.reset()
```

Details

This command disconnects all devices connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

[tspnet.connect\(\)](#) (on page 7-229)

tspnet.termination()

This function sets the device line termination sequence.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

<i>type</i>	An enumerated value indicating the termination type: <ul style="list-style-type: none"> • <code>tspnet.TERM_LF</code> • <code>tspnet.TERM_CR</code> • <code>tspnet.TERM_CRLF</code> • <code>tspnet.TERM_LFCR</code>
<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>termSequence</i>	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.

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`.

The termination sequence resets to the default value when the connection is terminated.

Example

```
myID = tspnet.connect("192.0.2.1")
if myID then
    tspnet.termination(myID, tspnet.TERM_LF)
    tspnet.disconnect(myID)
end
```

Also see

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.disconnect\(\)](#) (on page 7-230)

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)	Yes	System reset	Create configuration script	20.0

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

<code>value</code>	The timeout duration in seconds (0.01 s to 30.0 s)
--------------------	----------------------------------------------------

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 contain fractional seconds, 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 7-229)

[tspnet.execute\(\)](#) (on page 7-230)

[tspnet.read\(\)](#) (on page 7-232)

tspnet.tsp.abort()

This function stops remote instrument execution.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.tsp.abort(connectionID)
```

<code>connectionID</code>	Integer value used as a handle for other tspnet commands
---------------------------	----------------------------------------------------------

Details

Sends an abort command to the remote instrument.

The specified connection must be valid and available.

Example

```
tspnet.tsp.abort(myConnection)  Stops myConnection.
```

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	System reset	Create configuration script	Enabled

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

<i>value</i>	1 (enable) or 0 (disable)
--------------	---------------------------

Details

This setting determines if the instrument sends an abort message when it attempts to connect using `tspnet.connect` to a TSP-enabled instrument.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not issue an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP device 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 issued. See [Communication interfaces](#) (on page 4-4).

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 7-229)

tspnet.tsp.rhtablecopy()

This function copies a reading buffer synchronous table from a remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
array = tspnet.tsp.rhtablecopy(connectionID, name)
array = tspnet.tsp.rhtablecopy(connectionID, name, startIndex, endIndex)
```

<i>array</i>	A copy of the synchronous table
<i>connectionID</i>	Integer value used as a handle for other tspnet commands
<i>name</i>	The full name of the reading buffer name and synchronous table to copy
<i>startIndex</i>	Integer start value
<i>endIndex</i>	Integer end value

Details

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data.

startIndex and *endIndex* specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

This command is limited to transferring 50,000 readings at a time.

Example 1

```
t = tspnet.tsp.rhtablecopy(myconnection,
    'myremotebuffername.readings', 1, 3)
print(t[1], t[2], t[3])
```

Output:

```
4.5653423423e-1
4.5267523423e-1
4.5753543423e-1
```

Example 2

```
times = tspnet.tsp.rhtablecopy(mytspdevice,
    'myremotebuffername.timestamps', 1, 3)
print(times)
```

Output:

```
01/01/2008
10:10:10.0000013,01/01/2008
10:10:10.0000233,01/01/2008
10:10:10.0000576
```

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote 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)
```

<i>connectionID</i>	Integer value used as an identifier for other <code>tspnet</code> commands
<i>name</i>	The name that is assigned to the script
<i>script</i>	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

To load only and run at a later time, make sure the script contains only functions. Use `tspnet.execute()` to execute those functions at a later time.

If no name is specified, the script will be unnamed.

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`. The source of `mytest` will be the contents of the string passed for the third parameter.

Also see

[tspnet.execute\(\)](#) (on page 7-230)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.write(connectionID, inputString)
```

<i>connectionID</i>	The connection ID returned from <code>tspnet.connect()</code>
<i>inputString</i>	The string to be written

Details

The `tspnet.write()` function sends *inputString* to the remote instrument. It does not wait for command completion on the remote instrument.

The Model 707B or 708B 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.

Example

```
tspnet.write(myID, "runscript()\r\n")
```

Commands the remote instrument to run script named "myID".

Also see

[tspnet.connect\(\)](#) (on page 7-229)

[tspnet.disconnect\(\)](#) (on page 7-230)

[tspnet.read\(\)](#) (on page 7-232)

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)
```

<i>name</i>	The name of the string; the key of the key-value pair
<i>value</i>	The string to associate with <i>name</i> ; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

Example

```

userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")

```

Stores user-defined strings in nonvolatile memory.

Also see

[userstring.catalog\(\)](#) (on page 7-241)

[userstring.delete\(\)](#) (on page 7-242)

[userstring.get\(\)](#) (on page 7-242)

userstring.catalog()

This function creates an iterator for the user string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
for name in userstring.catalog() do ... end
```

name

The name of the string; the key of the key-value pair

Details

The catalog provides access for `userstring` 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 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 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 7-240)

[userstring.delete\(\)](#) (on page 7-242)

[userstring.get\(\)](#) (on page 7-242)

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)
```

<i>name</i>	The name (key) of the key-value pair of the <i>userstring</i> 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 7-240)

[userstring.catalog\(\)](#) (on page 7-241)

[userstring.get\(\)](#) (on page 7-242)

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)
```

<i>value</i>	The value of the <i>userstring</i> key-value pair
<i>name</i>	The name (key) of the <i>userstring</i>

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```
value = userstring.get("assetnumber")
print(value)
```

Output: 236

Also see

[userstring.add\(\)](#) (on page 7-240)

[userstring.catalog\(\)](#) (on page 7-241)

[userstring.delete\(\)](#) (on page 7-242)

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	Yes			

Usage

```
waitcomplete()
waitcomplete(group)
```

<i>group</i>	Specifies which TSP-Link group on which to wait
--------------	-------------------------------------------------

Details

This function will wait for all previously started overlapped commands to complete. Currently the Models 707B and 708B have no overlapped commands implemented. However, other TSP-enabled products like the Series 2600A SourceMeter Instruments have overlapped commands. Therefore, when the Model 707B or 708B 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 from the master node.

If no *group* is specified, the local group is used.

If 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

<code>waitcomplete()</code>	Waits for all nodes in the local group.
-----------------------------	-----------------------------------------

Example 2

<code>waitcomplete(G)</code>	Waits for all nodes in group G.
------------------------------	---------------------------------

Example 3

<code>waitcomplete(0)</code>	Waits for all nodes on the TSP-Link network.
------------------------------	----------------------------------------------

Also see

None

Troubleshooting guide

In this section:

Troubleshooting guide.....	8-1
Error and status messages	8-1
USB troubleshooting	8-2
Troubleshooting GPIB interfaces	8-5
Troubleshooting LAN interfaces.....	8-5
Testing the display, keys, and channel matrix.....	8-9
Update drivers.....	8-10
Contacting support.....	8-11

Troubleshooting guide

This section provides information to help you troubleshoot problems with your instrument.

Error and status messages

This section includes information on error levels and how to read errors.

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.

Error queue remote commands

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.

ICL commands associated with the error queue	
Command	Description
errorqueue.clear() (on page 7-85)	Clear error queue of all errors
errorqueue.count (on page 7-86)	Number of messages in the error queue
errorqueue.next() (on page 7-86)	Request next error message from queue

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.



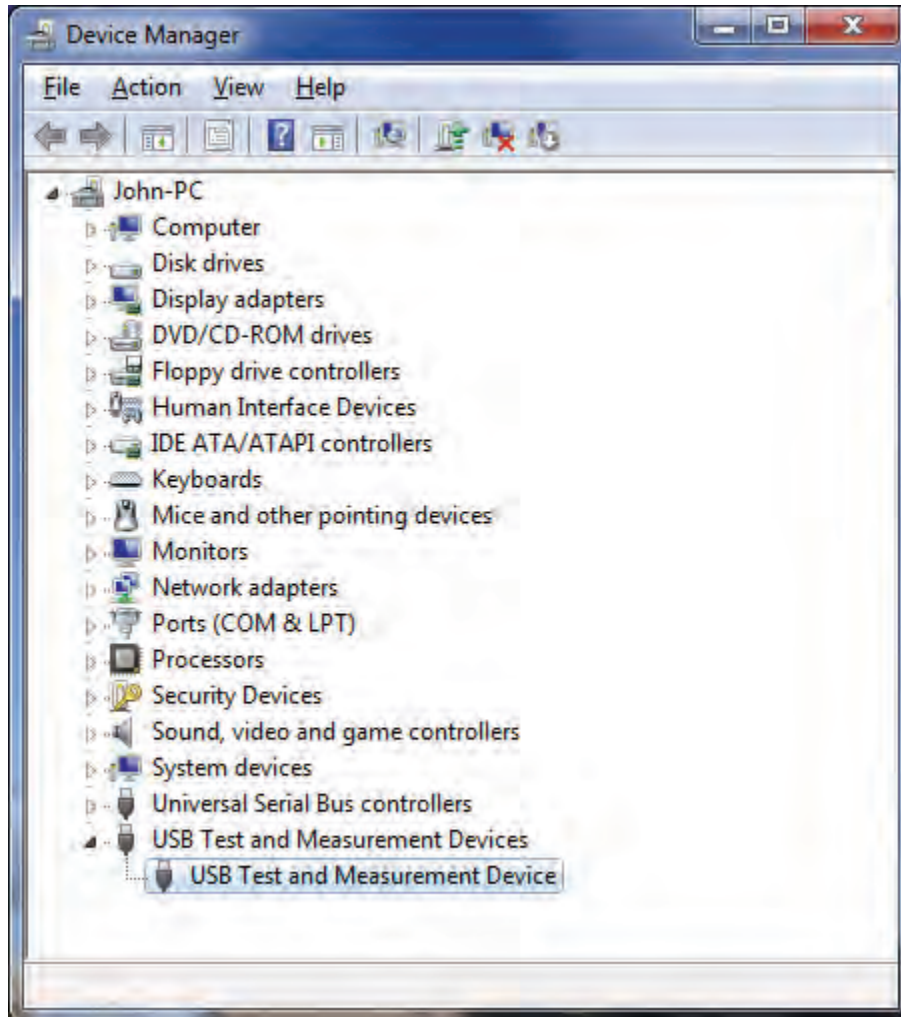
Quick Tip

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 93: 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 94: 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 2-64).

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 [Communication interfaces](#) (on page 4-4).

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) (<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 LXI conformant instruments 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.

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.

To reset the Model 707B or 708B, from the front panel, select **MENU**, then select **LAN > RESET**.

Install LXI Discovery Browser software on your computer

You can use the LXI Discovery Browser to identify the IP addresses of LXI certified instruments that are set up for automatic IP address selection. Once identified, you can double-click the IP address in the LXI Discovery Browser to open the web interface for the instrument.

The LXI Discovery Browser is available on the instrument CD. It is also available on the [Keithley website](http://www.keithley.com) (<http://www.keithley.com>).

To locate the LXI Discovery Browser on the website:

1. Select the **Support** tab.
2. In the model number box, type **707B** or **708B**.
3. From the list, select **Software**. A list of software applications for the Model 707B or 708B is displayed.
4. See the readme file included with the application for more information.

For more information on the LXI Consortium, see the [LXI Consortium website](http://www.lxistandard.org) (<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 2-25) 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 crosspoint display (707B only) 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 7-80).
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.

Verify crosspoint display operation (707B only)

You can verify that the LEDs and displays on the crosspoint display are working properly.

To verify crosspoint display operation:

1. From the front panel, select **MAIN MENU > DISPLAY > TEST > LED-PATTERNS**. The text "ALPH NUMERIC COL LEDS" is displayed.
2. Press the navigation wheel to display first test. The name of the test is displayed on the bottom display.
3. After each test, press the navigation wheel to move to the next test.
4. On the last tests (STEP COL GRIP LED WHEEL TO DIAL), use the navigation wheel to check the matrix LEDs and the slot LEDs.
5. 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](http://www.keithley.com/support) (<http://www.keithley.com/support>).

To see what drivers are available for your instrument:

1. Go to the [Keithley website](http://www.keithley.com) (<http://www.keithley.com>).
2. Select the Support tab.
3. Enter the model number of your instrument.
4. Select Software Driver from the list.

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

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) (<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 Model 707B or 708B 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 Model 707B or 708B serial number is displayed.

Locating information on the installed cards

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 card 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 for Model 707B or 1 for Model 708B)

Example

To get a list of all switching cards installed in the slots of a Model 707B, 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:

```
7174, 8x12 Fast Low-I Matrix, 01.00a, <Module Serial Number>
7072, 8x12 Semi Matrix, 01.00a, <Module Serial Number>
Empty Slot
Empty Slot
Empty Slot
Empty Slot
```

Frequently asked questions

In this section:

How do I get my LAN or web connection to work?	9-1
Why can't I close a channel?	9-1
How do I know if an error has occurred on my instrument?	9-2
How do I find the serial number and firmware version of the instrument?	9-3

How do I get my LAN or web connection to work?


For troubleshooting suggestions, see [Troubleshooting LAN interfaces](#) (on page 8-5).

For more detailed information on remote interface connections, see [Communication interfaces](#) (on page 4-4).

Why can't I close a channel?

The channel might be set to be forbidden to close.

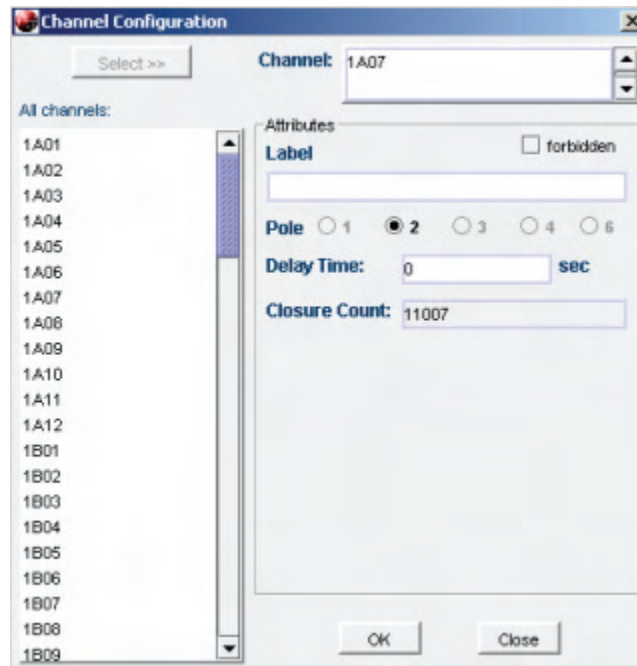
To check the forbidden state of a channel from the front panel:

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 95: 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 7-18)
- [channel.getforbidden\(\)](#) (on page 7-30)
- [channel.setforbidden\(\)](#) (on page 7-45)

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 7-86) and [errorqueue.next\(\)](#) (on page 7-86) 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 7-133) to 1 (enabled).

To set the instrument to automatically send prompts after each command message, set [localnode.prompts](#) (on page 7-129) 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 Model 707B or 708B 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 Model 707B or 708B serial number is displayed.

In this section:

Additional Models 707B and 708B information..... 10-1

Additional Models 707B and 708B information

For additional information on the Model 707B or 708B, refer to:

- The CD-ROM (ships with the product): Contains software tools, drivers, and product documentation, including documentation for switch cards that are compatible with the Models 707B and 708B
- The [Keithley Instruments website](http://www.keithley.com) (<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*
 - *Switching Handbook: A Guide to Signal Switching in Automated Test Systems*
 - Application notes
 - Updated drivers
 - Information on related products, including:
 - Switch cards, including the Models 7072, 7072-HV, 7173-50, and 7174A
 - The Model 4200-SCS Semiconductor Characterization System
 - The Series 2600A System SourceMeter Instruments
- Your local Field Applications Engineer can help you with product selection, configuration, and usage. Check the website for contact information.

Maintenance

In this appendix:

Upgrading firmware	A-1
Check fan status.....	A-2
Fuse replacement.....	A-2

Upgrading firmware

You can upgrade the instrument firmware from the web interface.

To upgrade the firmware:

1. From the left navigation area, select **Unit**.
2. Log in if necessary.
3. From the Unit buttons, click **Upgrade Firmware**.
4. A confirmation message is displayed. Click **OK**.
5. A version message is displayed. Select the appropriate option.
6. Locate the file.
7. 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.

8. To complete the upgrade, the instrument power must be cycled. If not done automatically at the end of the upgrade process, manually cycle instrument power.

NOTE

If you have a GPIB or USB connection, you can also use TSB to upgrade the firmware. See the TSB help files for information.

Check fan status

You can check the status of the fan from the front panel of the Model 707B.

In addition, if the fan is not operating on power up, the message "Failed to sense fan" is displayed.

To check the fan status:

1. From the front panel, select **MAIN MENU > UNIT-INFO > FAN**.
2. Press the navigation wheel. The status is displayed:
 - Fan Normal: Fan is operating normally.
 - Fan Failure: Fan is not moving.
3. Press **EXIT** to return to the menu.

If the fan is not operating, contact Keithley Instruments. See [Contacting support](#) (on page 8-11).

Fuse replacement

The fuses on the Model 707B or 708B are located on the rear panel of the instrument, as shown below.

Replacement fuses are listed in the following table.

Replacement fuses		
Model	Rating	Keithley Instruments part number
707B	250V, dual 2.0A slow blow	Two FU-106-2.0
708B	250V / 1.0A slow blow	One FU-72

WARNING

Disconnect all external power from the equipment and the line cord before performing any maintenance on the Model 707B or 708B.

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.

To replace a fuse:

1. Use a small flat-tip screwdriver to lift the tab at the bottom of the fuse module.
2. Pull the fuse module out.
3. Replace the fuse.
4. Re-install the fuse module.

If the fuse continues to blow, a circuit malfunction exists and must be corrected. See [Contacting support](#) (on page 8-11).

Figure 96: 707B fuse location

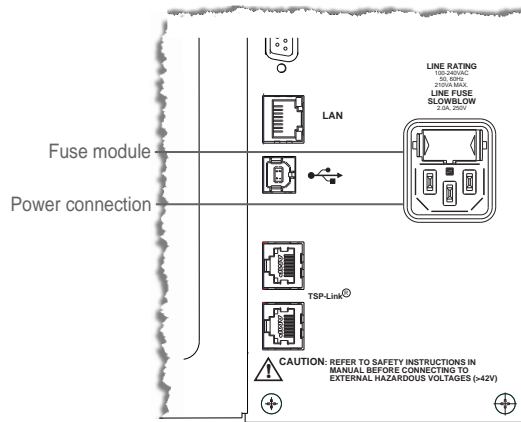
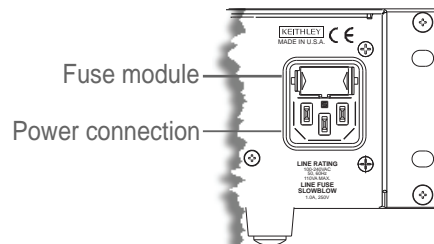


Figure 97: 708B fuse module location



Using Models 707A and 708A compatibility mode

In this appendix:

Model A to Model B differences.....	B-1
Models 707A and 708A commands.....	B-3

Model A to Model B differences

You can use a Model 707B or 708B in an existing Model 707A or 708A application. The units are compatible with the following exceptions:

- Master/subordinate operation is not supported.
- The digital I/O is limited to twelve bits for all models.
- Relay test is not supported.
- Some commands operate differently (see [Models 707A and 708A commands](#) (on page B-3)).

When using Model 707A or 708A compatibility mode, the only compatible remote interface is GPIB.

You can select one of two options when enabling Model 707A or 708A DDC compatibility mode:

- **70xA-VERSION:** This option most closely matches Model 707A or 708A operation. Use this version if you are transferring applications directly from a Model 707A or 708A with few changes.
- **70xB-VERSION:** This options provides enhanced operation, including error checking and more robust settling time operation. Use this version if you are updating existing applications.

To enable Model 707A or 708A DDC compatibility mode:

1. From the front panel, select **MENU**.
2. Select **DDC**.
3. Select **ENABLE**.
4. Select the version.
5. Press **ENTER**.

Cycle instrument power.

Front-panel relay closure indicators

When you are using Model 707A or 708A compatibility mode, the channel information is displayed on the updated display. For information on the display, see [Front-panel operation](#) (on page 2-11).

NOTE

Models 707B and 708B channel notation is different than on the Models 707A and 708A. For Models 707A and 708A, the slot number is built into the column number. For Models 707B and 708B, the slot number is the first number of the channel notation. For example, in Models 707A and 708A, crosspoint "A56" refers to slot 5, row A, column 8. This same crosspoint appears as "5A08" on the Models 707B and 708B front-panel display. However, with DDC emulation enabled, the Models 707B and 708B will accept "A56" and close the correct crosspoint.

Timing issues

The Models 707B and 708B run much faster than their predecessors. When using code from an older switch model, be aware that timing problems could be introduced into the system.

Digital interface

When you are using Model 707A or 708A compatibility mode, the digital I/O is fixed as follows:

- Digital input: Digital I/O lines 1 to 6
- Digital output: Digital I/O lines 7 to 12
- External trigger: Digital I/O line 13
- Matrix ready: Digital I/O line 14

Refer to [Digital I/O port](#) (on page 2-7) for the pinout diagram for the digital I/O connector.

Memory setups

Memory setups are handled as a single channel pattern when you use Model 707A or 708A compatibility mode.

The Model 707B or 708B supports 100 memory setups, in addition to the channel patterns normally available in the Model 707B or 708B.

The memory patterns are named MEMSETUP_{xxxx}, where _{xxxx} is between 001 and 100.

Memory patterns are created as they are used.

You can work with memory patterns through TSP as you do with channel patterns. See [Channel patterns](#) (on page 2-99).

Models 707A and 708A commands

Models 707A and 708A commands		
Command	Description	Differences in Models 707B and 708B
An	Edge for which an externally generated pulse executes a trigger	None.
Bn	Select logic sense of matrix ready	Actual matrix ready signal may have different timing characteristics. If a relay does not change state, the matrix ready signal does not include the relay settle time.
Crc(,rc)..(,rc)	Close crosspoints in a setup	Actual ready and matrix ready signals may have different timing characteristics. The Ready signal includes the relay settle time. Subordinate units are not supported. In the case of multiple commands, all crosspoints are closed simultaneously.
Dnnnnnn	Set text on the display	Only available on Model 707B. The display character limit has been increased to 20 and the text is displayed on the first line of the VFD display.
Db,s	Set digital output	Only available on Model 708B. The actual digital output settings will not function for bits that are outside of the physical interface specifications (see Digital interface (on page B-2)).
En	Specify setup number	None.
Fn	Enable or disable triggers	None.
Gn	Output format	None.
Hn	Front panel key	Only Model 707A. This command has no effect and does not issue an error message when used.
In	Insert blank setup	None.
Jn	Self-test	There is no self-test for the Models 707B and 708B.
Kn	EOI and hold off	Actual ready and matrix ready signals may have different timing characteristics. The Ready signal on the Model 707B or 708B includes the relay settle time. The Models 707A and 708A could hold off the GPIB bus on the specific 'X' character. To increase performance, the Models 707B and 708B use a message-based system. Therefore, the hold off has been changed to 'hold off on end of message'. When the 'X' is at the end of the message, the functionality is identical.
Lbbbb...X	Download setup	None.
Mn	SRQ and Serial Poll Byte	Performing a self-test or pressing a key does not clear the SRQ Ready bit.
Nrc(,rc)...(,rc)	Open crosspoint	Actual ready and matrix ready signals may have different timing characteristics. The ready signal on the Model 707B or 708B includes the relay settle time. Subordinate units are not supported. All crosspoints are opened. For example, for the command "NA1NA2NA3", all three crosspoints are opened. All crosspoints are opened simultaneously.
Ovvv	Digital output	The actual digital output settings will not function for bits that are outside the physical interface specifications (see Digital interface (on page B-2)).
Pn	Clear crosspoints	None.
Qn	Delete setup	None.
Rn	Restore defaults	None.
Sn	Programmable settle time	Works as intended in 70xB-VERSION. For 70xA-VERSION, settle time is always zero (0).

Tn	Trigger source	The trigger source 0/1 (GPIB Talk) is not available for the Models 707B and 708B. The factory default address for the Models 707B and 708B is 16. The trigger source 8/9 (front panel key) is not implemented.
Un	Instrument Config/Status	For U0, the last pressed key always reads 05. For U1, the Self-Test, PowerUp, and Master/Subordinate Loop bits always read zero. For U4, the number is always zero. For U5, the subordinates always read zero. For U7, though compatible bits are returned, the actual digital input settings read zero for those bits outside of the physical interface specifications (see digital input/output difference). For U8, the test relay is not supported and always reads 15.
Vabcdefgh	Make before break	None.
Wabcdefgh	Break before make	None.
<cmd>X	Execute	None.
Yn	Change line terminate	None.
Zm,n	Copy setup	None.
<space>	Model and version	Sending a <space> only in a message causes the unit to return the model and version number.
*idn?	Unit identification	For compatibility, the version number is A03 (version number of DDC compatibility, not the firmware version). For example, on the Model 707A, the return string is "707A03".

There is a limit of 64 commands per execution.

For more detail on the Model 707A and 708A commands, see the appropriate instruction manual:

- For the Model 707A: 707A-901-01 (A - Sep 1998)(Instruction).pdf
- For the Model 708A: 708A-901-01 (A - Sep 1998)(Instruction).pdf

These instruction manuals are available on the [Keithley website](http://www.keithley.com) (http://www.keithley.com).

Status model

In this appendix:

Overview	C-1
Status model diagrams.....	C-3
Status function summary.....	C-12
Clearing registers and queues.....	C-13
Startup state	C-13
Programming and reading registers	C-13
Status byte and service request (SRQ)	C-15
TSP-Link system status.....	C-19

Overview

Each Keithley Instruments Models 707B and 708B Switching Matrix 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. Commands included in your test program 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 eventually flow into the Status Byte Register. As a programmer, you are in full control of all enable registers, while the Switching Matrix has full control of all event status registers. In order for an event to be accounted for in the register's summary bit, first enable it by setting the appropriate bit in the applicable enable register. The entire status model is illustrated by 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). The register sets are structured in the following manner:

- Status Byte Register (byte containing the following bits)
 - Bit 0: Measurement summary register sets the Measurement Summary Bit (MSB)
 - Bit 1: System summary register sets the System Summary Bit (SSB)
 - Bit 2: Error/Event queue sets the Error Available bit (EAV)
 - Bit 3: Questionable summary register sets the Questionable Summary Bit (QSB)
 - Bit 4: Output queue sets the Message Available bit (MAV)
 - Bit 5: Event summary register sets the Event Summary Bit (ESB)
 - Bit 6: Master summary status register sets the Master Summary Status bit (MSS) or a Service request sets the RQS bit.
 - Bit 7: Operation summary register sets the Operation Summary Bit (OSB) (byte containing the following bit):
 - - Operation user summary register (sets the User bit of the Operation Summary register)

Status register sets

Typically, a status register set contains the following registers:

- **Condition** (.condition): a read-only register that constantly updates 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): a read-write register that when set to 1, specifies a negative transition (change from 1 to 0) sets the event register bit.
- **Positive transition register** (.ptr): a read-write register that when set to 1, specifies a positive transition (change from 0 to 1) sets the event register bit.

When an event occurs, and the appropriate NTR or PTR bit is set, the matching event register bit is set to 1. The event bit remains latched to 1 until the register is read or the status model is reset. When an event register bit is set and its corresponding enable bit is set, the output (summary bit) of the register will set to 1. This in turn sets a bit in a higher-level register 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 Switching Matrix uses an Output queue and an Error or Event queue. Response messages, such as those generated from print commands, are placed in the Output Queue. As programming errors and status messages occur, they are placed in the Error queue. When a queue contains data, it sets the appropriate summary bit of the Status Byte Register (EAV for the Error or Event queue; MAV for the Output queue).

The [Status Byte register overview](#) (on page C-4) shows how the two 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.

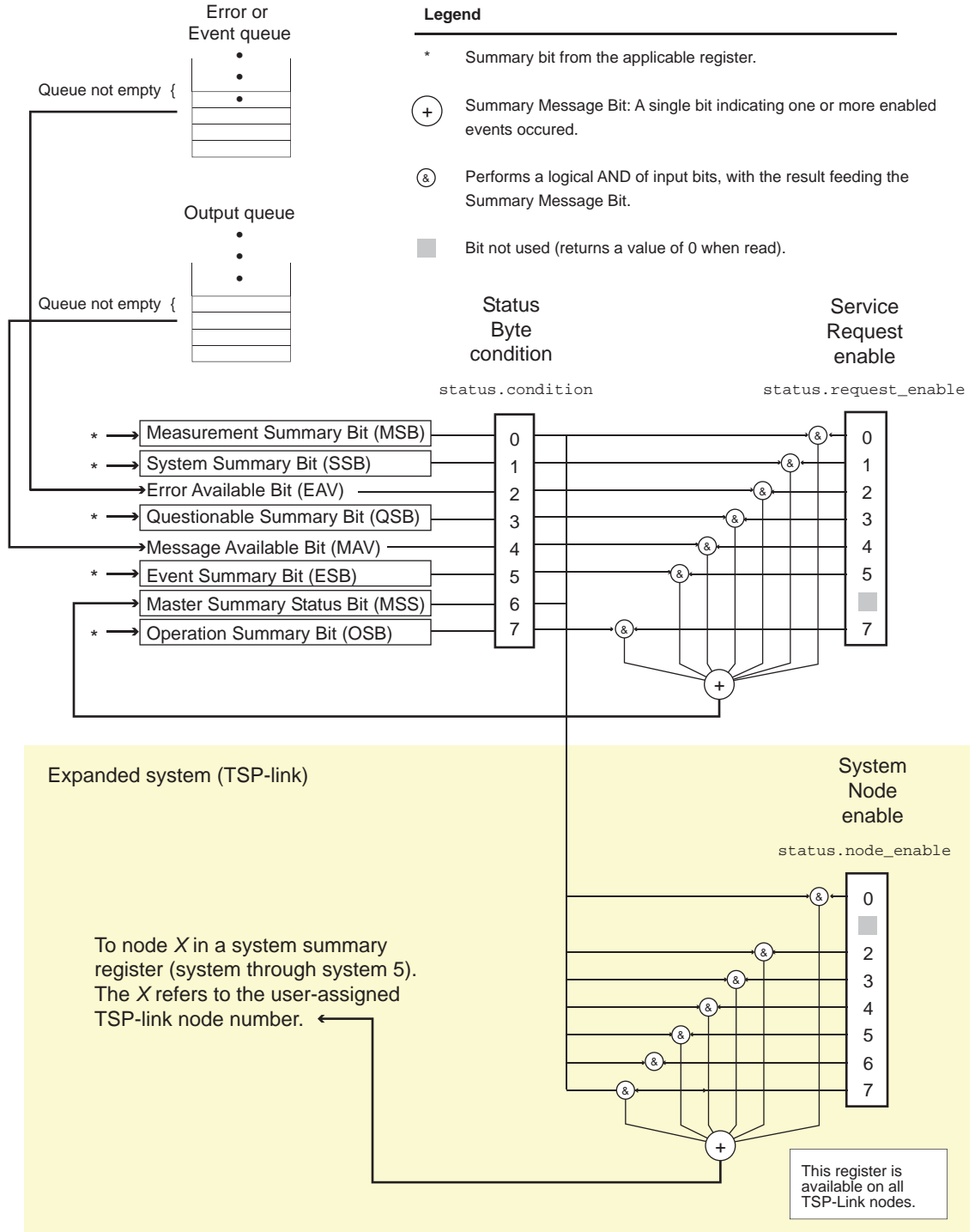
A message is read from the output queue by addressing the instrument to talk.

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 98: Status Byte register



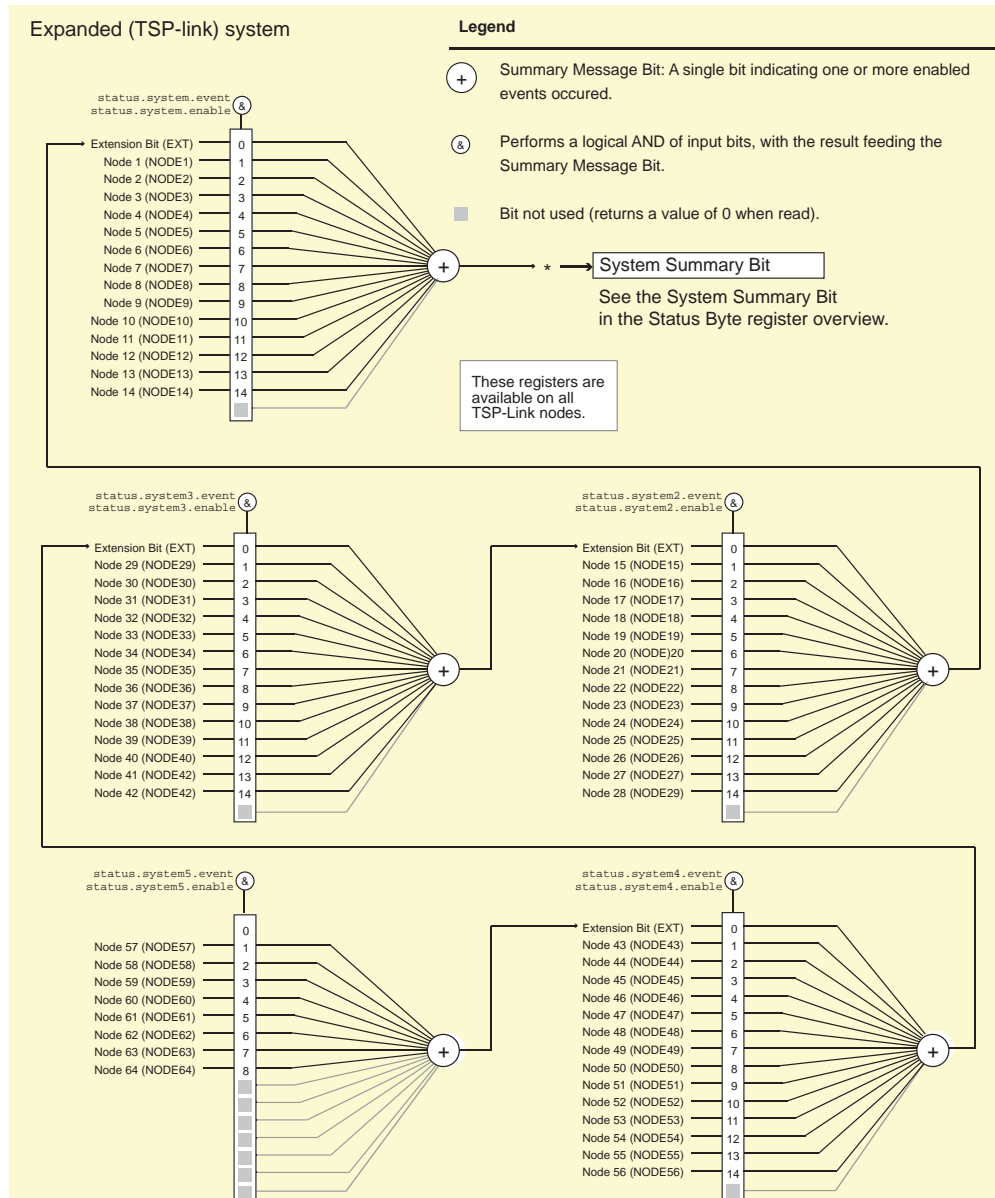
Measurement summary bit (Measurement event register)

The summary bit of the measurement event register provides enabled summary information to Bit 0 (MSB) of the status byte. Since the Models 707B and 708B Switching Matrix has no internal measurement capabilities, none of the bits in the measurement registers are defined. Therefore, this bit is always 0.

System summary bit (System register)

The summary bit of the system register provides enabled summary information to Bit 1 (SSB) of the status byte.

Figure 99: 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 C-19) for details on using the status model in a TSP-Link system.

Attributes are summarized in [status.system.*](#) (on page 7-190), [status.system2.*](#) (on page 7-192), [status.system3.*](#) (on page 7-194), [status.system4.*](#) (on page 7-196), and [status.system5.*](#) (on page 7-198).

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 C-14). 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* NODEN:** 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

Error available bit (Error or Event queue)

The summary bit of the Error or Event queue provides enabled summary information to Bit 2 (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 Switching Matrix 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
<code>errorqueue.clear()</code>	Clear error queue of all errors.
<code>errorqueue.count</code>	Number of messages in the error/event queue.
<code>errorqueue.next()</code>	Request error message.

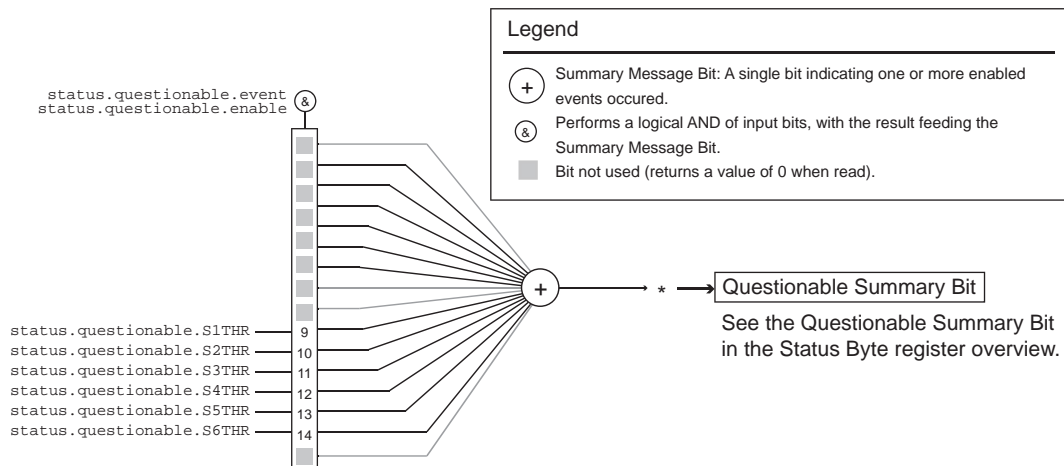
Messages in the Error queue include a code number, message text, severity, and TSP-Link node number. For example, the following commands request the next complete error information from the error queue and displays the code, message, severity and node of the next error:

```
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 3 (QSB) of the status byte.

Figure 100: Questionable summary bit (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 7-182). Keep in mind that bits can also be set by using numeric parameter values. For details, see [Programming enable and transition registers](#) (on page C-13).

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 4 (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 Switching Matrix queues associated with the status model. The other queue sets the [Error Available Bit \(Error or Event queue\)](#) (on page C-7). Both queues are first-in, first-out (FIFO) queues. The [Status Byte register overview](#) (on page C-4) shows how these queues are structured with regard to the other registers.

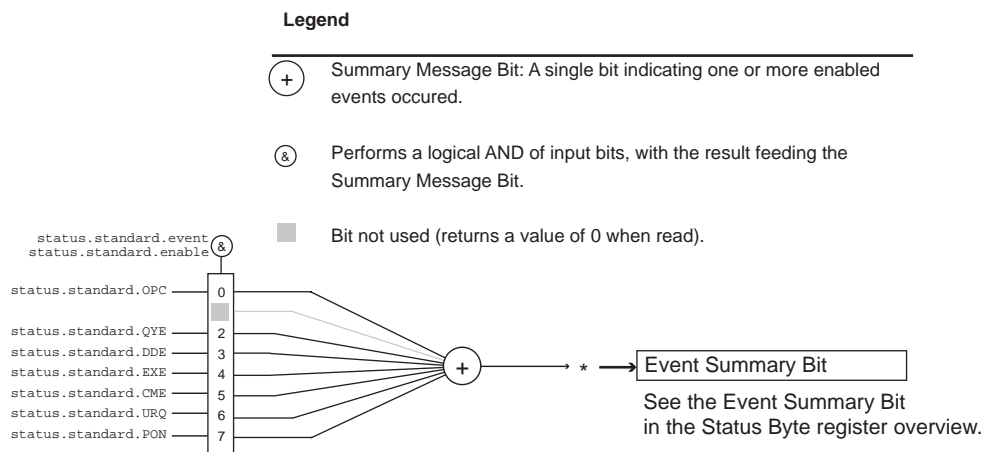
As an example, when a print command is sent, the response message is placed in the Output queue. When data is placed in the Output queue, the Message Available (MAV) bit in the Status Byte Register sets. A response message is cleared from the Output queue when it is read. The Output queue is considered 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 Switching Matrix to talk.

Event summary bit (ESB register)

The summary bit of the Standard event register provides enabled summary information to Bit 5 (OSB) of the status byte.

Figure 101: 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 7-188). Keep in mind that bits can also be set by using numeric parameter values. For details, see [Programming enable and transition registers](#) (on page C-13).

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 ICL function `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 6 (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).

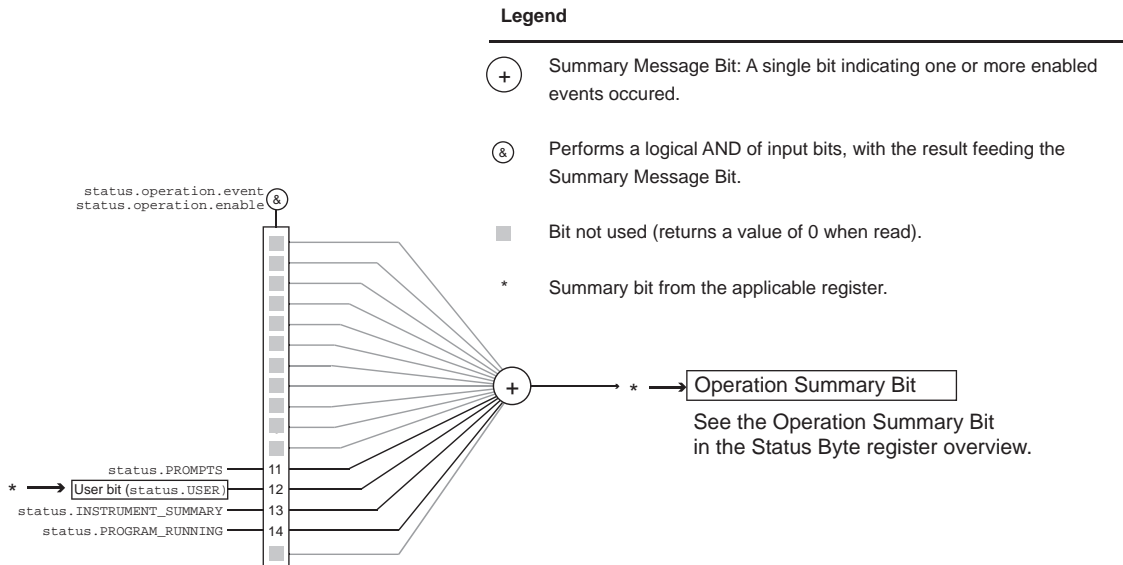
The Master Summary Status Bit (MSS) is set when an enabled summary bit of the Status Byte Register is set. This bit (B6) may also be interpreted as a Request Service (RQS) bit. Depending on how it is used, Bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit.

When using the GPIB serial poll sequence of the Switching Matrix to obtain the status byte (serial poll byte), B6 is the RQS bit. See [Serial polling and SRQ](#) (on page C-17) 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 will stay 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 7 (OSB) of the status byte.

Figure 102: Operation summary bit (Operation event register)



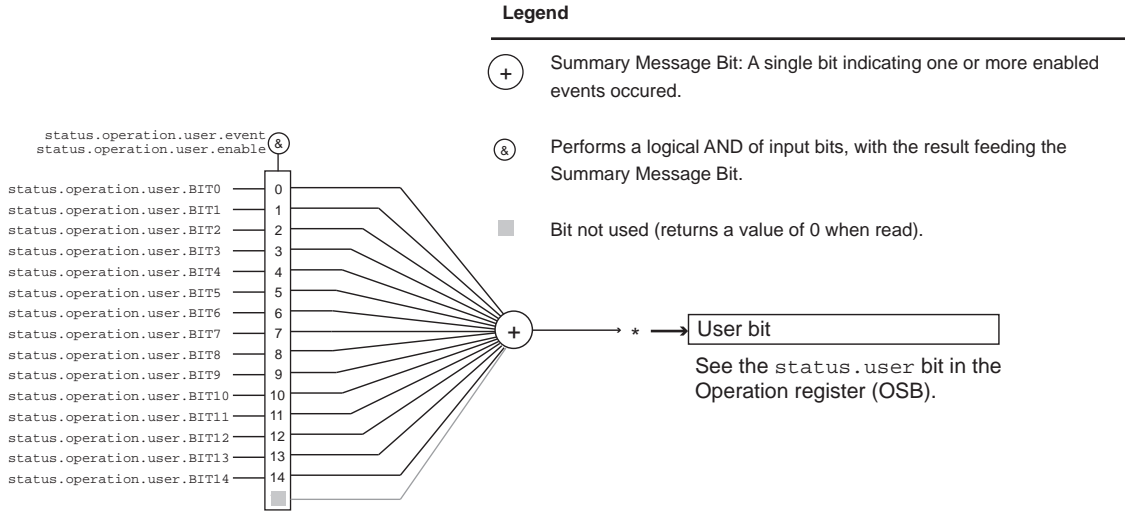
The bits used in this register set are described as follows:

- **Bits B1-B10:** Not used.
- **Bit B11, Remote Summary (REM):** Set bit indicates that an enabled in the Operation Status Remote Summary Register is set.
- **Bit B12, User (USER):** Set bit indicates that an enabled bit in the operation status user register is set.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that an enabled bit in the operation status instrument summary register is set.
- **Bit B14:** Not used.

Operation user bit (Operation user register)

The summary bit of the operation user register provides the user bit (User) (Bit 12) to the operation status register. In turn, the summary bit of the operation status register will provide the operation summary bit (OSB) (Bit 7) to the status byte.

Figure 103: 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 7-173) status.node_event (on page 7-177) status.node_enable (on page 7-175) status.request_event (on page 7-186) status.request_enable (on page 7-184) status.reset() (on page 7-187)
Operation event	status.operation.* (on page 7-178) status.operation.user.* (on page 7-180)
Questionable event	status.questionable.* (on page 7-182)
Standard event	status.standard.* (on page 7-188)
System events	status.system.* (on page 7-190) status.system2.* (on page 7-192) status.system3.* (on page 7-194) status.system4.* (on page 7-196) status.system5.* (on page 7-198)

Clearing registers and queues

Commands to reset the status registers and the Error queue are listed in the table below. In addition to these commands, any programmable register can be reset by sending the individual command to program the register with a 0 as its parameter value.

Commands to reset registers and clear queues

Commands	Description
To Reset Registers: *CLS <code>status.reset()</code>	Clears the output queue. Reset bits of status registers to 0. Reset bits of status registers to 0.
To Clear Error Queue: <code>errorqueue.clear()</code>	Clear all messages from Error Queue.

The instrument automatically clears the output queue when the instrument transitions from the local control state to the remote control state.

Startup state

When the Switching Matrix is turned on, various register status elements will be set as follows:

- The PON bit in the `status.operation.condition` register will be set.
- Other bits will be set appropriately based on the system's power-on configuration.
- All enable registers (`.enable`) will be set to 0.
- All negative transition registers (`.ntr`) will be set to 0.
- All used positive transition registers (`.ptr`) bits will be set to 1.
- The two queues will be empty.

Programming and reading registers

Programming enable and transition registers

The only registers that can be programmed by the user are the enable and transition registers. All other registers in the status structure are read-only registers. The following explains how to determine the parameter values for the various commands used to program enable registers. The actual commands are summarized in Common commands and Status function summary.

Figure 104: 16-bit status register

Bit Position	B7	B6	B5	B4	B3	B2	B1	B0
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2 ⁷)	(2 ⁶)	(2 ⁵)	(2 ⁴)	(2 ³)	(2 ²)	(2 ¹)	(2 ⁰)

A. Bits 0 through 7

Bit Position	B15	B14	B13	B12	B11	B10	B9	B8
Binary Value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32768	16384	8192	4096	2048	1024	512	256
Weights	(2 ¹⁵)	(2 ¹⁴)	(2 ¹³)	(2 ¹²)	(2 ¹¹)	(2 ¹⁰)	(2 ⁹)	(2 ⁸)

B. Bits 8 through 15

When using a numeric parameter, registers are programmed by including the appropriate `<mask>` value, for example:

```
*ese 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 returns a numeric value, while the `print(tostring())` command returns the string equivalent. For example, any of the following commands requests the Service Request Enable register value:

```
*SRE?
print(tostring(status.request_enable))
print(status.request_enable)
```

The response message will be a decimal value that indicates which bits in the register are set. That value can be converted to its binary equivalent using the information in [Programming enable and transition registers](#) (on page C-13). 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 an SRQ and set the system summary bit in all TSP-Link nodes when the thermal aspect bit of a card in slot 1 is set:

```
-- 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)

Two 8-bit registers control Service request: the Status Byte Register and the Service Request Enable Register. The [Status byte register](#) (on page C-16) describes the structure of these registers.

Service request enable register

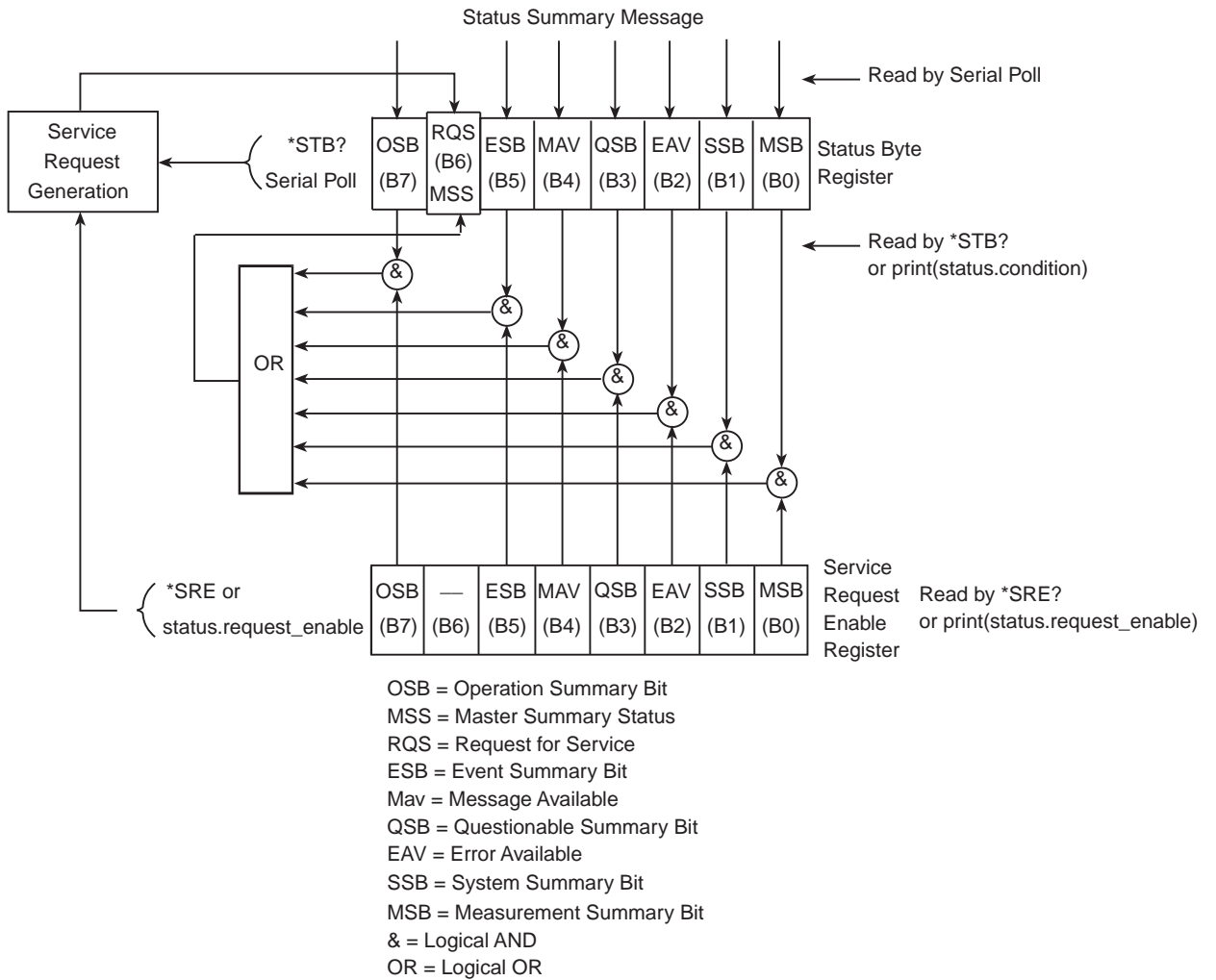
The Service Request Enable Register controls the generation of a service request. This register is programmed by the user and is used to enable or disable the setting of Bit B6 (RQS/MSS) by the Status Summary Message bits (B0, B1, B2, B3, B4, B5, and B7) of the Status Byte Register. As shown in [Status byte register](#) (on page C-16), 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 C-18).

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.

Figure 105: Status byte and service request (SRQ)



The bits of the Status Byte Register are described as follows:

- **Bit B0, Measurement Summary Bit (MSB):** Set summary bit indicates that an enabled measurement event has occurred.
- **Bit B1, System Summary Bit (SSB):** Set summary bit indicates that an enabled system event has occurred.
- **Bit B2, Error Available (EAV):** Set bit indicates that an error or status message is present in the Error Queue.
- **Bit B3, Questionable Summary Bit (QSB):** Set summary bit indicates that an enabled questionable event has occurred.
- **Bit B4, Message Available (MAV):** Set bit indicates that a response message is present in the Output Queue.
- **Bit B5, Event Summary Bit (ESB):** Set summary bit indicates that an enabled standard event has occurred.
- **Bit B6, Request Service (RQS)/Master Summary Status (MSS):** Set bit indicates that an enabled summary bit of the Status Byte Register is set. Depending on how it is used, Bit B6 of the Status Byte Register is either the Request for Service (RQS) bit or the Master Summary Status (MSS) bit:
 - When using the GPIB serial poll sequence of the Models 707B and 708B to obtain the status byte (serial poll byte), B6 is the RQS bit. See [Serial polling and SRQ](#) (on page C-17) 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 C-18) 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 will set bit B6 and generate 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 Switching Matrix. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register will remain cleared, and the program will simply proceed normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register will set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte register. This allows subsequent serial polls to monitor Bit B6 for an SRQ occurrence generated by other event types.

For common and script commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear MSS. The MSS bit stays set until all Status Byte 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 Switching Matrix. 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 (SRQs) and connections

SRQs affect both the GPIB and the VXI-11 connections. On a GPIB, the SRQ line will be asserted. On a VXI-11 connection, an SRQ event will be 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 C-18). 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 C-13) and [Reading registers](#) (on page C-14).

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
<code>*STB?</code> or <code>print(status.condition)</code>	Read the Status Byte register.
<code>*SRE <mask></code> or <code>status.request.enable = <mask></code>	Program the Service Request Enable register where <code><mask> = 0-255</code> .
<code>*SRE?</code> or <code>print(status.request.enable)</code>	Read the Service Request Enable register.

Enable and transition registers

In general, there are three types of user-writable registers that are used to configure which bits feed the register summary bit and when it occurs. The registers are identified in each applicable Instrument Control Library command table and defined as follows:

- **Enable register** (identified as `.enable` in each attributes 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 SRQ enable bits and read associated registers are summarized in the Status register sets which are summarized in the [Status Byte register overview](#) (on page C-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

The TSP-Link is an expansion interface that allows the instruments to communicate with each other. The test system can be expanded to include up to 32 TSP-Link 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 and running parallel test scripts](#) (on page 6-45) for details on the TSP-Link.

The system summary registers, shown in [Status Byte register overview](#) (on page C-4) and [System summary bit \(System register\)](#) (on page C-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 C-15) 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 thermal aspect event occurs on Node 15, the following sequence of events will occur:

1. On Node 15, with Bit B9 of the Questionable event register enabled, when the thermal aspect event occurs, Bit B9 bit sets (`status.questionable.condition`) which causes Bit B9 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.S1THR
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
```

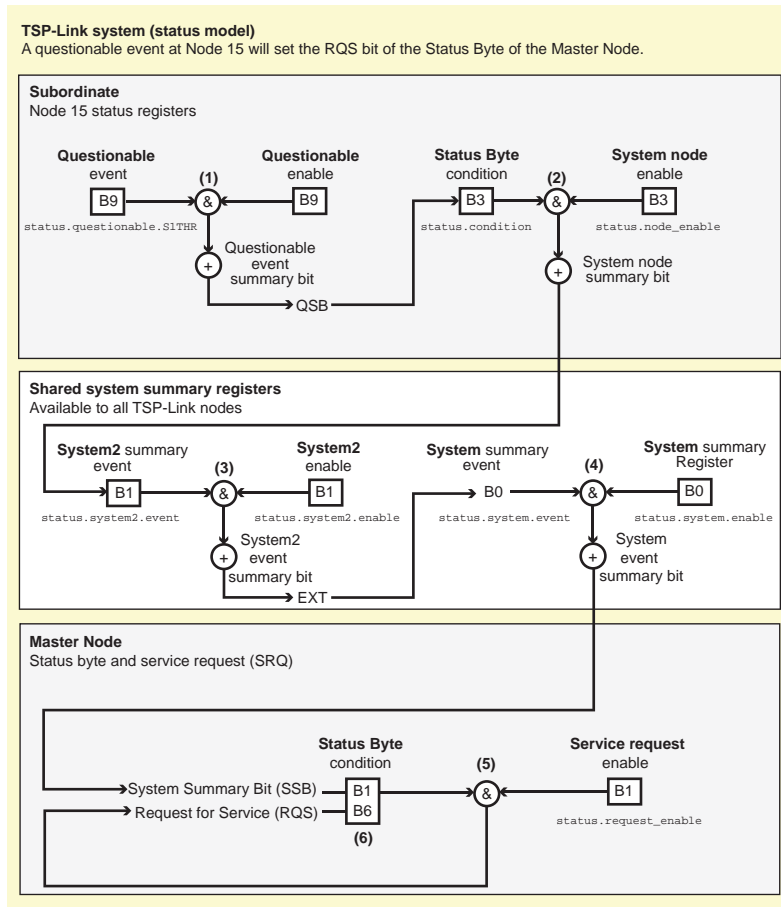
The affected system registers for the above commands are indicated by labels (3) and (4) (see the "TSP-Link status model configuration example" figure below).

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 106: TSP-Link status model configuration example



Index

<

<ch_list> queries • 2-89

A

acceptor trigger mode • 3-15, 3-19

annunciators • 7-67

anonymous script • 6-7

Arrays

arrays, TSL • 6-29

assigning groups • 6-51

attribute, assigning a value to • 5-2

attributes • 5-2

reading • 5-3

Autoexec script • 6-8

autorun scripts • 6-7

B

background scan execution • 3-7

base library functions • 6-30

basic scan procedure • 3-5

beeper • 5-4, 7-9

beeper functions and attributes

beeper.enable • 7-9

bit • See index

Boolean value • 7-16

field • 7-13

index • 7-11, 7-12, 7-13, 7-14, 7-15, 7-16

toggle • 7-17

weighted value, bit • 7-12

bit functions • 7-9, 7-10, 7-11, 7-12, 7-13, 7-14, 7-15, 7-16, 7-17, 7-67

bit.bitand() • 7-9

bit.bitor() • 7-10

bit.bitxor • 7-11

bit.clear() • 7-11

bit.get() • 7-12

bit.getfield() • 7-13

bit.set() • 7-14

bit.setfield() • 7-15

bit.test() • 7-16

bit.toggle() • 7-17

bitwise • 7-9, 7-10, 7-11

branching • 6-24

Break-Before-Make • 2-89

bus operation

scanning • 3-8

C

channel

break before make • 5-28, 7-20

- close/open operations and commands • 2-95, 7-19, 7-20
- connect rule • 2-89
- designations • 2-87
- existing scan • 3-6
- forbidden to close • 7-18, 7-45
- list parameter • 2-89
- make before break • 5-28, 7-20
- patterns • 2-99
- channel commands
 - channel.clearforbidden() • 7-18
 - channel.close() • 7-19
 - channel.connectrule • 7-20
 - channel.createspecifier() • 7-23
 - channel.getclose() • 7-27
 - channel.getcount() • 7-28
 - channel.getforbidden() • 7-30
 - channel.getlabelcolumn() • 7-32
 - channel.getlabelrow() • 7-33
 - channel.getstate() • 7-34
 - channel.gettype() • 7-35
 - channel.open() • 7-35
 - channel.pattern.catalog() • 7-37
 - channel.pattern.delete() • 7-38
 - channel.pattern.getimage() • 7-38
 - channel.pattern.setimage() • 7-39
 - channel.pattern.snapshot() • 7-41
 - channel.reset() • 7-43
 - channel.setdelay() • 7-44
 - channel.setforbidden() • 7-45
 - channel.setlabel() • 7-46
 - channel.setlabelcolumn() • 7-47
 - channel.setlabelrow() • 7-49
- channel patterns • 2-99
- clear • 7-67, 7-228
 - bit.clear() • See bit.clear()
- close
 - close channel operations • 2-95, 7-19, 7-24, 7-25, 7-27, 7-35
- command
 - Command programming
 - Time and date values • 7-3
 - programming notes • 7-1
 - queries • 5-3
- configuration
 - CONFIG key • 2-19
 - createconfigscript() • 7-50
 - script • See createconfigscript()
- connecting multiple units • 6-45, 6-46, 6-49
- connection
 - connect rule (channel) • 7-20
 - methods • 2-89
- contact information • 1-1
- counts • 3-4
- createconfigscript() • 7-50

crosspoint • 2-15

display (crosspoint) • 2-16, 2-17

cursor • 7-68, 7-81

D

delay functions

delay() • 7-55

digio functions and attributes

digio.readbit() • 7-55

digio.readport() • 7-56

digio.trigger[N].assert() • 7-57

digio.trigger[N].clear() • 7-57

digio.trigger[N].EVENT_ID • 7-58

digio.trigger[N].mode • 7-58

digio.trigger[N].overrun • 7-60

digio.trigger[N].pulsewidth • 7-60

digio.trigger[N].release() • 7-61

digio.trigger[N].reset() • 7-62

digio.trigger[N].stimulus • 7-62

digio.trigger[N].wait() • 7-64

digio.writebit() • 7-64

digio.writeprotect • 7-66

digio.writeport() • 7-65

Digital I/O

lines • 2-8

port • 2-7

Digital I/O port

+5V output • 2-8

Bit weighting • 2-9

Display

DISPLAY key • 2-18

Model 707B front panel • 2-12

Model 708B front panel • 2-12

display functions and attributes

display.(clear) • 7-67

display.getannunciators() • 7-67

display.getcursor() • 7-68

display.getlastkey() • 7-69

display.gettext() • 7-70

display.inputvalue() • 7-73

display.loadmenu.add() • 7-74

display.loadmenu.catalog() • 7-76

display.loadmenu.delete() • 7-76, 7-77

display.menu() • 7-77

display.prompt() • 7-78

display.screen • 7-80

display.sendkey() • 7-80

display.setcursor() • 7-81

display.settext() • 7-82

display.trigger.EVENT_ID • 7-84

display.waitkey() • 7-84

DMM

attributes, existing scan • 3-6

E

either edge trigger mode • 3-16

errorqueue functions and attribute

errorqueue.clear() • 7-85

errorqueue.count • 7-86

errors

effects on scripts • 8-2

summary • 8-1

Ethernet connector (RJ-45) • 2-4

event blenders • 3-22

eventlog functions and attributes

eventlog.all() • 7-87

eventlog.clear() • 7-88

eventlog.count • 7-89

eventlog.enable • 7-90

eventlog.next() • 7-90

eventlog.overwritemethod • 7-91

events • 3-21

examples

passing parameter • 5-3

script • 6-57

variable assignment • 5-3

exit functions

exit() • 7-92

F

falling edge trigger mode • 3-14

foreground scan execution • 3-7

format attributes

format.asciiprecision • 7-93

format.byteorder • 7-93

format.data • 7-94

front panel

display • 2-15, 2-18, 2-20

keys • 2-18, 2-19

Model 707B front panel • 2-12

Model 708B front panel • 2-12

scanning • 3-6

functions • 6-19

G

get • 7-12, 7-13, 7-27, 7-28, 7-29, 7-30, 7-31, 7-32, 7-33, 7-34, 7-35, 7-38, 7-67, 7-68, 7-69, 7-70, 7-96, 7-128

gpib attribute

gpib.address • 7-97

groups

assigning • 6-51

coordinating remote • 6-53

different test scripts • 6-50

leader • 6-51

reassigning • 6-51

H

hot switching • 1-1, 7-20

I

ICL

general device control • 6-59

ICL commands • 3-8

TSP-specific device control • 6-59

index • 7-11, 7-12, 7-13, 7-14, 7-15, 7-16

K

keys

 DISPLAY • 2-18

key-value pairs • 5-17, 7-240, 7-241, 7-242

L

LAN

 status light • 2-20

LAN functions and attributes

 lan.applysettings() • 7-98

 lan.config.dns. • 7-101

 lan.config.dns.address[N] • 7-98

 lan.config.dns.domain • 7-99

 lan.config.dns.dynamic • 7-100

 lan.config.dns.gateway • 7-102

 lan.config.dns.hostname • 7-101

 lan.config.ipaddress • 7-104

 lan.config.method • 7-104

 lan.config.subnetmask • 7-105

 lan.lxidomain • 7-106

 lan.nagle • 7-106

 lan.reset() • 7-107

 lan.restoredefaults() • 7-107

 lan.status.dns.address[N] • 7-108

 lan.status.dns.name • 7-109

 lan.status.duplex • 7-109

 lan.status.gateway • 7-110

 lan.status.ipaddress • 7-110

 lan.status.macaddress • 7-111

 lan.status.port.dst • 7-111

 lan.status.port.rawsocket • 7-112

 lan.status.port.telnet • 7-112

 lan.status.port.vxi11 • 7-113

 lan.status.speed • 7-113

 lan.status.subnetmask • 7-114

 lan.trigger[N].assert() • 7-114

 lan.trigger[N].clear() • 7-115

 lan.trigger[N].connect() • 7-116

 lan.trigger[N].connected • 7-117

 lan.trigger[N].disconnect() • 7-118

 lan.trigger[N].EVENT_ID • 7-118

 lan.trigger[N].ipaddress • 7-119

 lan.trigger[N].mode • 7-119

 lan.trigger[N].overrun • 7-121

 lan.trigger[N].protocol • 7-122

 lan.trigger[N].pseudostate • 7-122

 lan.trigger[N].stimulus • 7-123

 lan.trigger[N].wait() • 7-125

libraries, standard • 6-30

local group • 7-140

localnode functions and attributes

 localnode.define* • 7-125

 localnode.description • 7-126

 localnode.execute() • 7-127

- localnode.getglobal() • 7-128
- localnode.gettimezone() • 7-96
- localnode.model • 7-128
- localnode.password • 7-129
- localnode.prompts • 7-129
- localnode.prompts4882 • 7-130
- localnode.reset() • 7-131
- localnode.revision • 7-131
- localnode.serialno • 7-132
- localnode.setglobal() • 7-133
- localnode.settime() • 7-167
- localnode.settimezone() • 7-168
- localnode.showerrors • 7-133
- logical
 - logical AND operation • 7-9
 - logical OR operation • 7-10
- loop control • 6-25
- M**
- Make-Before-Break • 2-89
- makegetter functions
 - makegetter() • 7-134
 - makesetter() • 7-135
- master node • 7-140
- master node overview • 6-50
- master trigger mode, rising edge • 3-15, 3-18
- math
 - library functions • 6-33
- matrix card notation • 2-88
- Memory functions
 - memory.available() • 7-135
 - memory.used() • 7-137
- modules
 - identify installed • 2-93
- multiple units, connecting • 6-45, 6-46, 6-49
- N**
- named scripts
 - overview • 6-4
 - RUN • 6-7
- navigation wheel • 2-18
- node
 - master overview • 6-50
- nonblocking • 7-67, 7-81, 7-82
- O**
- open
 - open channel operations • 2-95
- overlapped operations in remote groups, coordinating • 6-53
- overwrite a bit field • 7-15
- P**
- parallel test scripts • 6-52
- power
 - blinking • 7-68
- POWER switch • 2-18
- precautions • 1-1

Precedence • 6-23

print functions

print() • 7-138

printbuffer() • 7-138

printnumber() • 7-140

programming

interaction • 6-36

script model • 6-3

pseudocards • 2-106

Q

queries • 5-3

queues • C-3

Output • C-3

R

reading buffer

removing stale values • 6-54

readings

errors • 8-2

registers

Enable and transition • C-19

Programming example • C-15

Reading • C-14

Serial polling and SRQ • C-17

Service request enable • C-15

relay closure count • 2-93

Reset

digio.trigger[N].reset() • 7-62

lan.reset() • 7-107

localnode.reset() • 7-131

reset() • 7-140

scan.reset() • 7-150

status.reset() • 7-187

timer.reset() • 7-201

rising edge

acceptor trigger mode • 3-15

master trigger mode • 3-15

trigger mode • 3-15

RJ-45

Ethernet connector • 2-4

run-time environment

script, restoring • 6-41

S

safety precautions • 1-1

Scan functions and attributes

scan.abort() • 7-141

scan.add() • 7-142

scan.addimagestep() • 7-143

scan.background() • 7-144

scan.bypass • 7-145

scan.create() • 7-146

scan.execute() • 7-147

scan.list() • 7-148

scan.mode • 7-150

scan.reset() • 7-150

- scan.scancount • 7-151
 - scan.state() • 7-152
 - scan.stepcount • 7-153
 - scan.trigger.arm.clear() • 7-153
 - scan.trigger.arm.set() • 7-154
 - scan.trigger.arm.stimulus • 7-154
 - scan.trigger.channel.clear() • 7-156
 - scan.trigger.channel.set() • 7-156
 - scan.trigger.channel.stimulus • 7-157
 - scan.trigger.clear() • 7-158
- scanning
- counts • 3-4
 - execution, foreground and background • 3-7
 - fundamentals • 3-1
- Script Editor • 6-36
- Script functions and attributes
- script.anonymous.name • 7-159
 - script.delete() • 7-159
 - script.new() • 7-160
 - script.restore() • 7-162
 - script.run() • 7-166
 - script.user.catalog() • 7-163
 - scriptVar.autorun • 7-163
 - scriptVariable.list() • 7-164
 - scriptVariable.name • 7-164
 - scriptVariable.save() • 7-166
 - scriptVariable.source • 7-167
- scripts
- autoexec • 6-8
 - autorun scripts • 6-7
 - deleting • 6-43
 - error effects • 8-2
 - examples • 6-57
 - function, using • 6-21
 - interactive • 6-3
 - name attribute • 7-164
 - named • 6-4, 6-7
 - parallel test, running • 6-52
 - restoring in run-time environment • 6-41
 - running • 6-6, 6-7, 6-52
 - Script Editor • 6-36
 - test scripts across the TSP-Link network • 6-54
 - unnamed • 6-7
 - user • 6-4, 6-6, 6-10
- scripts, error effects • 8-2
- serial polling • C-17
- slot
- slot list description • 2-16
- slot[X] attributes
- slot[X].idn • 7-169
 - slot[X].poles.four • 7-170
 - slot[X].poles.one • 7-171
 - slot[X].poles.two • 7-171
 - slot[X].pseudocard • 7-172

- sound • 7-9
- SRQ (Service Request) • C-15
- standard
 - libraries • 6-30
- standard libraries • 6-30
- state • 6-54
- status
 - LAN status indicator • 2-20
 - Status byte and service request (SRQ) • C-15
 - Commands • C-18
 - status functions and attributes
 - status. • 7-175
 - status.condition • 7-173
 - status.node_event • 7-177
 - status.operation.* • 7-178
 - status.operation.user.* • 7-180
 - status.questionable.* • 7-182
 - status.request_enable • 7-184
 - status.request_event • 7-186
 - status.reset() • 7-187
 - status.standard.* • 7-188
 - status.system.* • 7-190
 - status.system2.* • 7-192
 - status.system3.* • 7-194
 - status.system4.* • 7-196
 - status.system5.* • 7-198
 - status model • C-1
 - Clearing registers and queues • C-13
 - Programming registers and queues • C-13
 - Queues • C-3
 - Status byte and SRQ • C-2, C-15
 - Status register sets • C-2
 - TSP-Link system • C-19
 - status register sets • C-2
 - step counts • 3-4
 - string library functions • 6-31
 - style • 7-68, 7-81
 - substring • 6-31
 - synchronous
 - acceptor trigger mode • 3-19
 - master trigger mode • 3-18
 - trigger mode • 3-20
 - triggering modes, understanding • 3-17

T

 - tables • 6-29
 - test scripts across the TSP-Link network • 6-54
 - time • 7-167, 7-168
 - Timer functions
 - timer.measure.t() • 7-200
 - timer.reset() • 7-201
 - Trigger functions and attributes
 - trigger.blender[N].EVENT_ID • 7-202
 - trigger.blender[N].clear() • 7-201
 - trigger.blender[N].orenable • 7-202

- trigger.blender[N].overrun • 7-203
- trigger.blender[N].reset() • 7-203
- trigger.blender[N].stimulus[M] • 7-204
- trigger.blender[N].wait() • 7-205
- trigger.clear() • 7-206
- trigger.EVENT_ID • 7-206
- trigger.timer[N].clear() • 7-207
- trigger.timer[N].count • 7-207
- trigger.timer[N].delay • 7-208
- trigger.timer[N].delaylist • 7-209
- trigger.timer[N].EVENT_ID • 7-209
- trigger.timer[N].overrun • 7-210
- trigger.timer[N].passthrough • 7-211
- trigger.timer[N].reset() • 7-211
- trigger.timer[N].stimulus • 7-212
- trigger.timer[N].wait() • 7-213
- trigger.wait() • 7-214
- trigger mode • 3-15, 3-19
 - access trigger mode • 3-15, 3-19
 - either edge • 3-16
 - falling edge • 3-14
 - rising edge acceptor • 3-15
 - rising edge master • 3-15
 - synchronous • 3-20
 - synchronous acceptor • 3-19
 - synchronous master • 3-18
- trigger model
 - components • 3-3
 - described • 3-1
- TSP
 - advanced features • 6-45
 - installing software • 6-35
- tsplink function and attributes
 - tsplink.group • 7-214
 - tsplink.master • 7-215
 - tsplink.node • 7-215
 - tsplink.readbit() • 7-216
 - tsplink.readport() • 7-216
 - tsplink.reset() • 7-217
 - tsplink.state • 7-218
 - tsplink.trigger[N].assert() • 7-218
 - tsplink.trigger[N].clear() • 7-219
 - tsplink.trigger[N].EVENT_ID • 7-219
 - tsplink.trigger[N].mode • 7-220
 - tsplink.trigger[N].overrun • 7-221
 - tsplink.trigger[N].pulsewidth • 7-222
 - tsplink.trigger[N].release() • 7-223
 - tsplink.trigger[N].reset() • 7-223
 - tsplink.trigger[N].stimulus • 7-224
 - tsplink.trigger[N].wait() • 7-225
 - tsplink.writebit() • 7-226
 - tsplink.writeport() • 7-226
 - tsplink.writeprotect • 7-227
- TSP-LinkTM • 6-49

communicating between TSP-enabled instruments • 6-58

Initialization • 6-47

Node numbers • 6-48

Reset • 6-48

tspnet functions and attributes

tspnet.clear() • 7-228

tspnet.connect() • 7-229

tspnet.disconnect() • 7-230

tspnet.execute() • 7-230

tspnet.idn() • 7-232

tspnet.read() • 7-232

tspnet.readavailable() • 7-233

tspnet.reset() • 7-234

tspnet.termination() • 7-235

tspnet.timeout • 7-236

tspnet.tsp.abort() • 7-236

tspnet.tsp.abortonconnect • 7-237

tspnet.tsp.rhtablecopy() • 7-238

tspnet.tsp.runscript() • 7-239

tspnet.write() • 7-240

U

unnamed scripts • 6-7

user scripts

creating alternative • 6-4

modifying • 6-10

running • 6-6

userstring functions • 6-50

userstring.add() • 7-240

userstring.catalog() • 7-241

userstring.delete() • 7-242

userstring.get() • 7-242

UTC • 7-167

V

variables • 6-17

W

waitcomplete functions

waitcomplete() • 7-243

Warranty

Limitation of Warranty

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

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

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



A GREATER MEASURE OF CONFIDENCE

Keithley Instruments, Inc.

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

Specifications are subject to change without notice.
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.
All other trademarks and trade names are the property of their respective companies.



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

Keithley Instruments, Inc.

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • www.keithley.com