Tektronix®

MP5000 Series

Modular Precision Test System

Programmer Manual

Register now!

Click the following link to protect your product.

www.tek.com/register

077-1895-04 August 2026

These are the original instructions in English.

Go to www.tek.com/en/eula to read the Tektronix End User License Agreement.

Contacting Tektronix

Tektronix, Inc.
13725 SW Karl Braun Drive
P.O. Box 500
Beaverton, OR 97077
USA

For product information, sales, service, and technical support:

In North America, call 1-800-833-9200.

Worldwide, visit www.tek.com to find contacts in your area.

Contents

Safety precautions

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

This product is intended for use by 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

Production of this equipment required the extraction and use of natural resources. The equipment may contain substances that could be harmful to the environment or human health if improperly handled at the product's end of life. To avoid release of such substances into the environment and to reduce the use of natural resources, we encourage you to recycle this product in an appropriate system that will ensure that most of the materials are reused or recycled appropriately.

The  symbol indicates that this product complies with the applicable European Union requirements according to Directives 2012/19/EU and 2006/66/EC on waste electrical and electronic equipment (WEEE) and batteries. For information about recycling options, check the Tektronix Web site (www.tek.com/productrecycling).

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

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

This product may contain a small installed lithium metal button cell. Please properly dispose of or recycle the cell at its end of life according to local government regulations.

This product may contain one or more type CR lithium batteries. According to the state of California, CR lithium batteries are classified as perchlorate materials and require special handling. See http://www.dtsc.ca.gov/hazardouswaste/perchlorate for additional information.

If present, the small lithium primary button cell contained in this equipment does not exceed 1 gram of lithium metal content per cell, and the cell type has been shown by the manufacturer to comply with the applicable requirements of the UN Manual of Tests and Criteria Part III, Sub-section 38.3. Consult your carrier to determine which lithium battery transportation requirements are applicable to your configuration, including to its re-packaging and re-labeling, prior to reshipment of the product by any mode of transport.

Introduction

Welcome

This manual contains programming information for the MP5000 Series Modular Precision Test System and available modules.

The MP5000 Series mainframe and modules can be programmed using Test Script Processor (TSP™) commands using a remote interface such as a USB or a LAN connection. The Tektronix TSP™ Toolkit extension allows you to develop and edit scripts to create a customized source and measurement test system.

For higher channel count configurations, TSP-Link allows you to scale your test system in a master-subordinate, node-based configuration with multiple mainframes and other TSP-enabled instruments. TSP-Net allows the use of instruments without TSP-Link, allowing for an even more flexible test system.

Contact information

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

Organization of manual sections

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

- **Introduction to TSP operation:** Describes the basics of using Test Script Processor (TSP™) commands to control the instrument.
- **Configuration lists and sweeps:** Contains details on creating source and measure settings for a module channel and information on sweep operation, including programming examples.
- **Trigger models:** Information on triggering options, including trigger events, objects, timers, and trigger model blocks.
- **Reading buffers:** Contains details on reading buffer configuration, access, and content viewing.
- **TSP-Link and TSP-Net:** Describes TSP-Link™, a high-speed trigger synchronization and communications bus that you can use to connect multiple TSP-enabled instruments, and TSP-Net ™, which allows instruments without TSP capability to communicate in a TSP-Link test environment.
- **TSP command set reference tables by model:** Available commands for the MP5000 Series and modules, listed by model.
- **TSP command references:** Contains programming notes and listings of all TSP commands available for the MP5000 Series and modules.
- **Trigger model TSP commands:** Contains the TSP commands used to build a trigger model flow for your test system.
- **Common commands:** Contains descriptions of IEEE Std 488.2 common commands.
- **Status model:** Includes status model details for the MP5103 Series and modules.

Overview of available documentation

The documentation available for the MP5000 Series Modular Precision Test System is available from tek.com. The documentation includes:

- **User Manual:** Provides initial setup instructions, installation, a description of the instrument controls and connections, basic maintenance, and operational validation procedures.
- **Programmer Manual:** Contains command descriptions for controlling the MP5103 and available modules as well as information on building trigger models, TSP-Link and TSP-Net, and reading buffers.
- **Reference Manuals:** Include advanced operation topics, hardware configuration, additional maintenance information, troubleshooting procedures, and guidelines for measurement considerations and optimization.
- **TSP Toolkit Quick Start Guide:** Provides instructions for the Tektronix TSP™ Toolkit extension for Microsoft™ Visual Studio Code™.
- **Code examples for the MP5000 Series and modules:** Tektronix GitHub Programmatic Control Examples Repository.
- **Accessories information:** Documentation for accessories available for the MP5103 and modules.

In addition to documentation, the website also provides the latest instrument firmware and additional support information.

TSP Toolkit software

The Tektronix TSP™ Toolkit is a Microsoft™ Visual Studio Code™ extension that provides support for the Tektronix Test Script Processor (TSP) technology to edit and execute scripts on TSP-enabled Tektronix instruments.

The extension includes language features such as syntax error detection, code navigation, and code-completion suggestions, as well as `.tsp` command set documentation and hover help.

You can download TSP Toolkit from the Microsoft Visual Studio Code Marketplace.

Enable remote communications

The MP5103 provides the following features for securing remote communications:

- Communications protocol disable and enable
- Optional password requirement for remote access

Enable USBTMC

USBTMC is enabled by default and cannot be disabled. If LAN is disabled or protected when no password is set, you must use USBTMC for remote communications or to set up LAN communications remotely.

USBTMC can be optionally password-protected. This requires authorization using the `login` command before communications are accepted.

To set USBTMC to be password protected:

Send

```
usbtmc.access = localnode.PROTECT
```

To remove password protection from USBTMC:

Send

```
usbtmc.access = localnode.ENABLE
```

Enable LAN

By default, all LAN communications are disabled. When they are enabled, they are enabled in protected mode, which requires a password.

Set the password before using remote communications with LAN. LAN access requires login by default, but this can be disabled for use without security.

NOTE: The web browser interface for the MP5103 always requires a password.

When using password protection to secure LAN communications, it is recommended that you use a strong password that contains the following:

- A minimum of 8 characters
- A mixture of lowercase letters, uppercase letters, numbers, or the accepted special characters and symbols `! @ # $ % ^ & *`

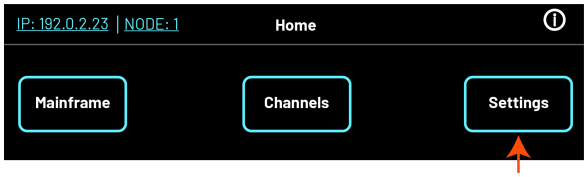
Set the mainframe password

Passwords must be 1 to 20 characters long. Only alphanumeric characters and the special characters `! @ # $ % ^ & *` are permitted.

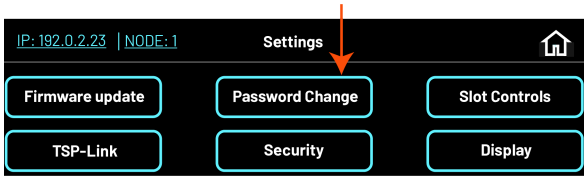
The default, fixed user name is `admin`.

To set the password from the front panel of the mainframe:

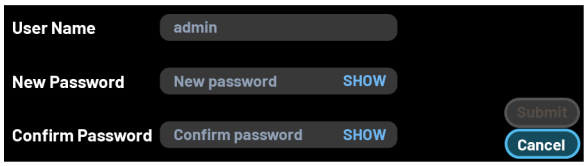
1. From the Home screen, select **Settings**.



2. Select **Password Change**.



3. Enter and confirm the new password using the on-screen keyboard.



4. Select **Submit**.

To set the password using remote commands:

Send the following command:

```
user.password.change("newpassword")
```

Where *newpassword* is your new password.

Enable and disable LAN communications

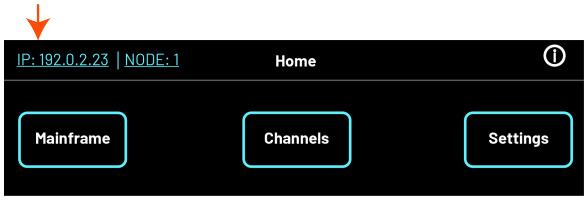
You can enable or disable a protocol. When you disable a protocol, external users are prevented from establishing a connection and communications through a specified interface or port.

Password protection can be enabled for LAN protocols. When password protection is enabled, a user must log in to the mainframe with the set password before the connection is accepted.

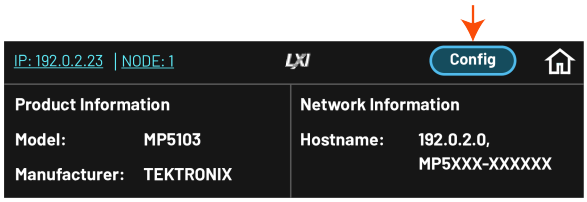
NOTE: By default, all LAN protocols are disabled. To enable LAN protocols, use the primary LAN access attribute `lan.enable` or the setting from the front panel. LAN must be enabled before individual protocols can be disabled.

To enable the LAN protocol from the front panel:

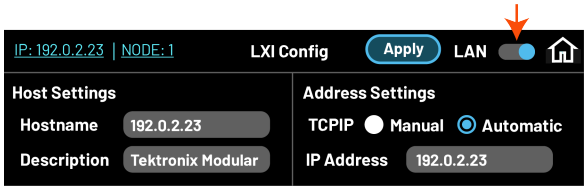
1. From the Home screen, select the IP address of your instrument.



2. Select **Config**.



3. Enable LAN with the slider. You can also configure additional LAN settings.



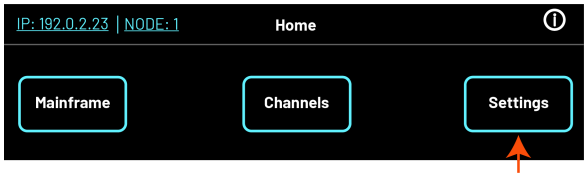
4. Select **Submit** to apply the settings.

Adjust LAN protocol and access settings

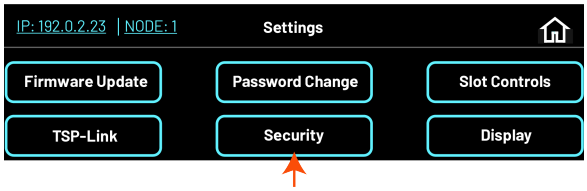
After enabling LAN communications, you can enable, disable, or password-protect LAN protocols.

To adjust protocol settings from the front panel:

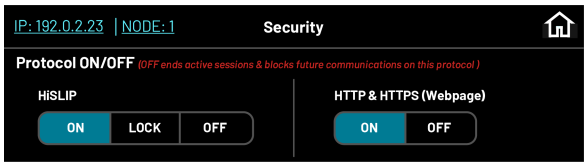
1. From the Home screen, select **Settings**.



2. Select **Security**.

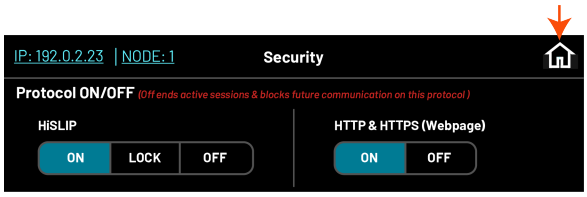


3. Available protocols and options are displayed. Turn a protocol on, off, or select **LOCK** to enable password protection.



NOTE: When password protection is enabled for a protocol, authorization using the `login` command is required before communications are accepted.

4. Select **Home** to exit.



To adjust LAN protocol settings with remote commands:

You can use the following TSP command attributes to enable, disable, or protect a protocol or access:

- **Enable:** Permit communications without a password
- **Disable:** Turn off communications
- **Protect:** Require authorization using the `login` command before communications are accepted

Protocol / Access	Attribute	Options
LAN (All)	`lan.enable`	`localnode.ENABLE` `localnode.DISABLE`

HiSLIP	`lan.hislip.access`	`localnode.ENABLE`
		`localnode.PROTECT`
		`localnode.DISABLE`

Raw sockets	`lan.rawsocket.access`	`localnode.ENABLE`
		`localnode.PROTECT`
		`localnode.DISABLE`

Telnet	`lan.telnet.access`	`localnode.ENABLE`
		`localnode.PROTECT`
		`localnode.DISABLE`

Network ping request	`lan.ping.enable`	`localnode.ENABLE` `localnode.DISABLE`
Multicast DNS	`lan.mdns.enable`	`localnode.ENABLE` `localnode.DISABLE`
Webpage	`lan.web.enable`	`localnode.ENABLE` `localnode.DISABLE`

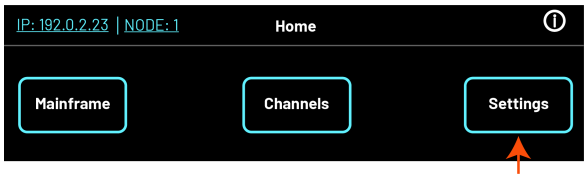
To adjust LAN protocol settings from the web interface:

1. Select **Settings**.
2. Select **Security**.
3. Available protocols and options are displayed. Turn a protocol on, off, or select **LOCK** to enable password protection.

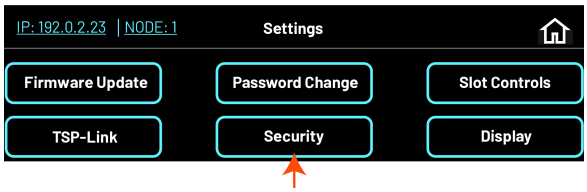
Example: Enable HiSLIP with password protection

To enable HiSLIP with password protection from the front panel:

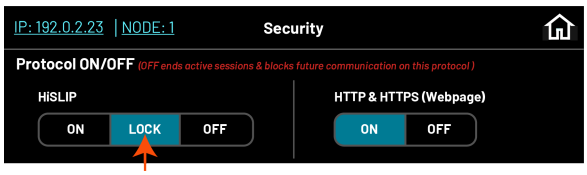
1. Select **Settings**.



2. Select **Security**.



3. Available protocols and options are displayed. Select **LOCK** for the HiSLIP protocol.



To enable HiSLIP with password protection with remote commands:

Send the following:

```
lan.enable = localnode.ENABLE
lan.hislip.access = localnode.PROTECT
```

To enable HiSLIP with password protection with the web interface:

1. Select **Settings**.
2. Select **Security**.
3. Select **LOCK** for the HiSLIP protocol.

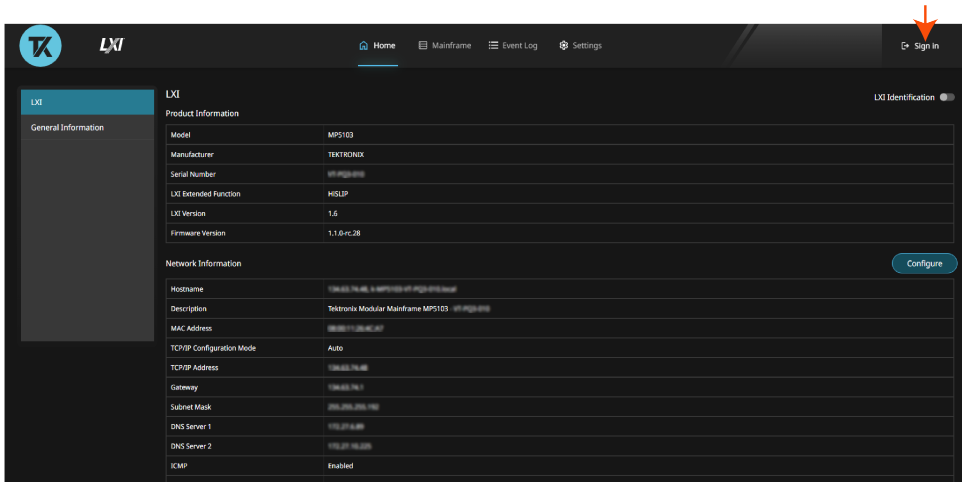
Log in to the MP5103

If the protocol you want to use is password-protected, you must log in before communicating with the MP5103.

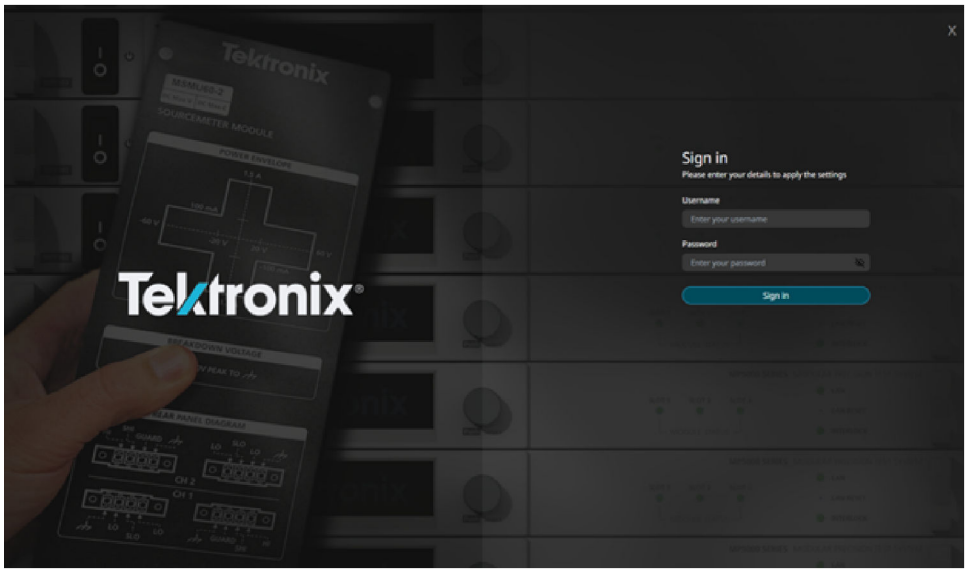
NOTE: The web interface for the MP5103 always requires a password.

To log in to the web browser interface:

1. Enter the IP address of your instrument into a web browser.
2. Select **Sign in** on the top-right of the browser window.



3. Enter your username and password, then select **Sign in**.



To log in using remote commands:

Send

```
login password
```

Where *password* is your LAN access password.

Log out of the MP5103

To log out of the MP5103 using the web interface, close your browser.

To log out of the MP5103 using remote commands, send:

```
logout
```

Communications troubleshooting

The following topics can help to diagnose communications issues. A password may be enabled for the communications protocol you are using or another user may be communicating with the mainframe.

The password can be cleared or reset from the front panel or with a remote command.

Query the LAN password authentication state

To determine if a password is enabled for the communications protocol you are using, send

`login?`

Responses indicate whether password authorization is required before the mainframe responds to remote input.

`Authenticated`	A password has been set and the user is logged in. Communications are enabled for the selected protocol.
`Unauthenticated`	The protocol is password-protected and the user is not logged in. Send `login` *`password`*, where *`password`* is your access password.

This command is useful for diagnosing communications issues when the mainframe does not respond to remote input, such as TSP commands. Sending `login?` can eliminate password protection as a cause for timeouts.

Query the mainframe communication status

Only one communication protocol can be used at a time. Before using a new protocol, the previous one must be closed by sending `abort`. Most communication libraries do this when the connection is terminated.

To determine if another protocol is in use on the mainframe:

1. Connect using an interface.
2. Send the `remote?` command. Responses indicate whether your command interface is allowed to control the mainframe:

`Allowed`	The mainframe is in remote mode and ready to receive commands on the selected protocol. Other protocols are ignored.
`Disallowed`	Another protocol is in use. Commands on this connection are ignored.
`Local`	The mainframe is in local mode. All remote connections are terminated.

This command is useful for diagnosing issues if the mainframe does not respond to commands. If the mainframe does not respond, it may be because another user is connected through a different protocol or actively using the same protocol. Communications are not allowed until the other connection is closed by sending the `abort` command.

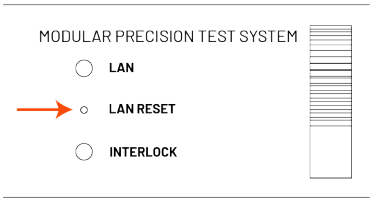
Clear or reset the password

NOTE: These procedures reset the password and disable LAN communications.

After clearing or resetting the password, follow the instructions in Set the mainframe password (on page 23) and Enable and disable LAN communications (on page 24).

To reset the LAN password from the front panel of the MP5103:

Using a straightened paper clip or a similar thin, rigid object, press and release the **LAN Reset** button on the front of the MP5103.



To reset the LAN password using a remote command:

Send:

```
lan.reset()
```

Introduction to TSP operation

Introduction to TSP operation

Instruments that are enabled for Test Script Processor (TSP™) operate like conventional instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you do when using any other instrument.

Unlike conventional instruments, TSP-enabled instruments can execute automated test sequences independently, without an external controller. You can store these commands as a script that can be run later by sending a single command message to the instrument.

You do not have to choose between using conventional control or script control. You can combine these forms of instrument control in the way that works best for your test application.

Control the instrument by sending individual command messages

You can send a message that contains remote commands to control an instrument through the communications interface. You can use a test program that resides on a computer (the controller) to sequence the actions of the instrument.

TSP commands can be function-based or attribute-based. Function-based commands are commands that control actions or activities. Attribute-based commands define characteristics of an instrument feature or operation.

Constants represent fixed values.

Functions

Function-based commands control actions or activities. A function-based command performs an immediate action on the instrument.

Each function consists of a function name followed by a set of parentheses `( )`. Only include information in the parentheses if the function takes a parameter. If the function takes one or more parameters, they are placed between the parentheses and separated by commas.

Attributes

Attribute-based commands are commands that set the characteristics of an instrument feature or operation. For example, a characteristic of TSP-enabled instruments is the model number (`localnode.model`).

Attributes can be read-only, read-write, or write-only. They can be used as a parameter of a function or assigned to another variable.

Assigning a value to an attribute

To set the characteristics, attribute-based commands define a value. For many attributes, the value is in the form of a number, enumerated type, or a predefined constant.

Reading an attribute

To read an attribute, you can use the attribute as the parameter of a function, such as `print()`, or assign it to another variable.

Queries

Instruments enabled for Test Script Processor (TSP™) do not have inherent query commands. Like other scripting environments, the `print()` and `printnumber()` commands generate output in the form of response messages. Each `print()` command creates one response message.

Scripts and programs for TSP

Automated test programs or scripts consist of both sequences of commands and control code. At a basic level, the controlling computer sends commands to the instrument and receives the data. The controlling program then can make decisions on that data, process the data, and control further testing.

TSP-enabled instruments contain a scripting engine that is also a Lua interpreter. The native Lua programming language for this interpreter has been extended to include control commands specific to Tektronix instruments. This is the command set of the instrument.

Lua programming commands can also be sent and interpreted by the instrument. Programming commands control script execution and provide tools such as variables, functions, branching, and loop control.

Example of sending individual commands

This example shows how to use single TSP commands to control the instrument. Any module can be used to execute these commands.

```
-- Use a function to print a number out
printnumber(5)
-- Use an attribute to change the number of digits in a number
format.asciiprecision = 1
-- Use a function to print the same number
printnumber(5)
```

Output:

```
5.00000e+00
5e+00
```

The functions complete an action, such as printing something to the console, and the attribute changes a setting in the instrument.

Lua basics

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.

The following topics contain the basics about the Lua programming language to allow you to start adding Lua programming commands to your scripts.

For more information about Lua, see the Lua website. Another source of useful information is the Lua users group, created for and by users of Lua programming language.

Comments

You can start a comment anywhere outside a string by typing a double hyphen (`--`). If the text immediately after `--` is anything other than double left brackets (`[[`), the comment is a short comment, which continues until the end of the line.

If `--` is followed by `[[`, the following characters are a long comment, which continues until double right brackets (`]]`) close the comment. Long comments may continue for several lines and may contain nested `[[ . . . ]]` pairs.

An example of a short comment is:

```
-- Turn off the front-panel display.
```

An example of a long comment is:

```
--[[Display a menu with three menu items. If the second menu item is selected,
the selection is given the value Test2.]]
```

Function and variable name restrictions

Variable names must conform to the following rules:

- Start with an alphabetic character.
- Cannot contain periods.
- Cannot exist as a global variable, local variable, or array.

If you create a variable that has the same name as an existing variable, the existing variable is overwritten by the new variable.

You cannot use Lua reserved words and top-level command names for function or variable names.

You cannot use the following Lua reserved words in the following table 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`

You also cannot use top-level command names listed in the following table as variable names. If you use these names, it results in the loss of use of the commands. For example, if you send the command `digio = 5`, you cannot access the `digio.*` commands until you either run `restoreglobals` or turn the instrument power off and then turn it on again.

`bit`	`io`	`print`	`tonumber`
`collectgarbage`	`lan`	`printbuffer`	`tostring`
`dataqueue`	`localnode`	`printnumber`	`trigger`
`delay`	`login`	`reset`	`tsplink`
`digio`	`logout`	`script`	`tspnet`
`eventlog`	`makegetter`	`settime`	`usbtmc`
`exit`	`makesetter`	`settimezone`	`user`
`firmware`	`math`	`slot`	`userstring`
`format`	`node`	`status`	`waitcomplete`
`fs`	`opc`	`string`	
`gettimezone`	`os`	`timer`	

Shell keywords

The following commands are considered shell keywords. They affect the behavior of the runtime of the instrument. They can only be sent directly to the instrument using TSP Toolkit or another program that can send individual commands. These keywords cannot be used in scripts. They are also not equivalent to TSP functions and do not require parentheses.

Command	Description
`abort`	Aborts all processes
`login` *password*	Logs into the instrument on the communication interface being used. For usage, see Enable LAN (on page 23).

Command	Description
`login?`	Queries the LAN password-enabled state. For usage, see Query the LAN password state (on page 29).
`logout`	Logs out of the instrument on the communication interface being used. For usage, see Enable LAN (on page 23).
`flash`	Indicates the start of a firmware file being loaded to the mainframe. The mainframe will not execute commands until `endflash` is sent. See firmware.image.load() (on page 167).
`endflash`	Indicates the end of a firmware file being loaded to the mainframe. The update process should be started using firmware.update() (on page 168). See firmware.image.load() (on page 167).
`endscript`	Indicates the end of a script being loaded to the runtime. Commands received after this command are immediately executed. For usage, see Load a script (on page 45).
`loadandrunscript` *scriptName*	Indicates the start of a script being loaded. Commands received after this command are not immediately executed. Once `endscript` is received, the script is run. For usage, see Load a script (on page 45).
`loadscript` *scriptName*	Indicates start of script being loaded to the runtime. Commands received after this command are not immediately executed. For usage, see Load a script (on page 45).
`remote?`	Determines if another connection to the mainframe is active. For usage, see Query the mainframe user status (on page 29).

Values and variable types

In Lua, you use variables to store values in the runtime environment for later use.

Lua is a dynamically-typed language; the type of the variable is determined by the value that is assigned to the variable.

Variables in Lua are assumed to be global unless they are explicitly declared to be local. A global variable is accessible by all commands. Global variables do not exist until they have been assigned a value.

Delete a global variable

To delete a global variable, assign `nil` to the global variable. This removes the global variable from the runtime environment.

Functions

With Lua, you can group commands and statements using the `function` and `end` keywords. Functions can take zero, one, or multiple parameters, and they return zero, one, or multiple values.

You can use functions to form expressions that calculate and return a value. Functions can also act as statements that execute specific tasks.

Functions are first-class values in Lua. That means that functions can be stored in variables, passed as arguments to other functions, and returned as results. They can also be stored in tables.

When a function is defined, it is stored in the runtime environment. Like all data that is stored in the runtime environment, the function persists until it is removed from the runtime environment, is overwritten, or the instrument is turned off.

Use the function keyword to create functions

Functions are created with a message or in Lua code in either of the following forms:

```
function myFunction(parameterX) functionBody end
myFunction = function (parameterX) functionBody end
```

Where:

- *myFunction*: The name of the function.
- *parameterX*: Parameter names. To use multiple parameters, separate the names with commas.
- *functionBody*: The code that is executed when the function is called.

To execute a function, substitute appropriate values for *parameterX* and insert them into a message formatted as:

```
myFunction(valueForParameterX, valueForParameterY)
```

Where *valueForParameterX* and *valueForParameterY* represent the values to be passed to the function call for the given parameters.

NOTE: The output you get from these examples may vary depending on the data format settings of the instrument.

Example

```
function add_two(first_value, second_value)
   return first_value + second_value
end
print(add_two(3, 4))
```

Creates a variable named `add_two` that has a variable type of function.

Output:

```
7
```

Operators

You can compare and manipulate Lua variables and constants using operators.

Arithmetic operators

Operator	Description
+	addition
−	subtraction
*	multiplication
/	division
−	negation (for example, c = −a)
^	exponentiation

Relational operators

Operator	Description
<	less than
>	greater than
<=	less than or equal
>=	greater than or equal
~=	not equal
==	equal

Logical operators

Operator	Description
not	returns `false` or `true`
and	returns first argument if argument is `false` or `nil`, otherwise returns second argument
or	returns its second argument if different than `nil` or `false`, otherwise returns second argument

NOTE: 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.

String operators

Operator	Description
..	Concatenates two strings. If either argument is a number, it is coerced to a string (in a reasonable format) before concatenation.

Operator precedence

Operator precedence in Lua follows the order shown in the following table (from higher to lower priority).

Operator
^ (exponentiation)
not, - (unary)
*, /
+, -
.. (concatenation)
<, >, <=, >=, ~=, ==
and
or

You can use parentheses to change the precedences in an expression. The concatenation ("`..`") and exponentiation ("`^`") operators are right associative. All other binary operators are left associative. The following examples show 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.

In Lua, `nil` and `false` are `false` and everything else is `true`. Zero (`0`) is `true` in Lua.

The syntax of a conditional block is as follows:

```
if expression then
   block
elseif expression then
   block
else
   block
end
```

Where:

- *`expression`* is Lua code that evaluates to either `true` or `false`
- *`block`* consists of one or more Lua statements

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 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

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`

For loops

There are two variations of `for` statements supported in Lua: Numeric and generic.

```
for condition do
   block
end
```

Where:

- *`condition`* is either stepping through a numeric count or a generic iterator
- *`block`* is the body of the code loop

Break

The `break` statement terminates the execution of a `while`, `repeat`, or `for` loop, skipping to the next statement after the loop. A break ends the innermost enclosing loop.

Return and break statements can only be written as the last statement of a block. If it is necessary to return or break in the middle of a block, an explicit inner block can be used.

Tables and arrays

Lua makes extensive use of the data type table, which is a flexible array-like data type. Table indexes 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"]`.

Standard libraries

In addition to the standard programming constructs described in this document, Lua includes standard libraries that contain useful functions for string manipulation, mathematics, and related functions. Test Script Processor (TSP™) scripting engine instruments also include instrument control extension libraries, which provide programming interfaces to the instrumentation that can be accessed by the TSP scripting engine. These libraries are automatically loaded when the TSP scripting engine starts and do not need to be managed by the programmer.

The following topics provide information on some of the basic Lua standard libraries. For additional information, see the Lua website. The TSP scripting engine uses Lua 5.0.2.

Base library functions

Function	Description
`collectgarbage()` `collectgarbage(`*`limit`*`)`	Sets the garbage-collection threshold to the given limit (in kilobytes) and checks it against the byte counter. If the new threshold is smaller than the byte counter, Lua immediately runs the garbage collector. If there is no limit parameter, it defaults to zero (0), which forces a garbage-collection cycle. See Lua memory management (on page 39) for more information.
`gcinfo()`	Returns the number of kilobytes of dynamic memory that the Test Script Processor (TSP™) scripting engine is using and returns the present garbage collector threshold (also in kilobytes). See Lua memory management (on page 39) for more information.

Function	Description
`tonumber(x)` `tonumber(x, base)`	Returns *x* converted to a number. If *x* is already a number, or a convertible string, the number is returned; otherwise, it returns `nil`. An optional argument specifies the base to use when interpreting the numeral. The base may be any integer from 2 to 36, inclusive. In bases above 10, the letter `A` (in either upper or lower case) represents 10, `B` represents 11, and so forth, with `Z` representing 35. In base 10, the default, the number may have a decimal part and an optional exponent. In other bases, only unsigned integers are accepted.
`tostring(x)`	Receives an argument of any type and converts it to a string in a reasonable format.
`type(v)`	Returns the type of its only argument as a string. The possible results of this function are `"nil"` (a string, not the value `nil`), `"number"`, `"string"`, `"boolean"`, `"table"`, `"function"`, `"thread"`, and `"userdata"`.

Lua memory management

Lua automatically manages memory, which means you do not have to allocate memory for new objects and free it when the objects are no longer needed. Lua occasionally runs a garbage collector to collect all objects that are no longer accessible from Lua. All objects in Lua are subject to automatic management, including tables, variables, functions, threads, and strings.

Lua uses two numbers to control its garbage-collection cycles. One number counts how many bytes of dynamic memory Lua is using; the other is a threshold. When the number of bytes crosses the threshold, Lua runs the garbage collector, which reclaims the memory of all inaccessible objects. The byte counter is adjusted, and the threshold is reset to twice the new value of the byte counter.

String library functions

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

Function	Description

`string.byte(s)`
`string.byte(s, i)`
`string.byte(s, i, j)`
	Returns the internal numeric codes of the characters `s[i], s[i+1], ···, s[j]`. The default value for *i* is `1`; the default value for *j* is *i*.

`string.char(···)`	Receives zero or more integers separated by commas. Returns a string with length equal to the number of arguments, in which each character has the internal numeric code equal to its corresponding argument.

Function	Description

`string.format(`
`  formatstring, ...)`
	Returns a formatted version of its variable number of arguments following the description given in its first argument, which must be a string. The format string follows the same rules as the `printf` family of standard C functions. The only differences are that the modifiers `*`, `l`, `L`, `n`, `p`, and `h` are not supported and there is an extra option, `q`. The `q` option formats a string in a form suitable to be safely read back by the Lua interpreter; the string is written between double quotes, and all double quotes, newlines, embedded zeros, and backslashes in the string are correctly escaped when written.
	
	For example, the call:
	
	`string.format('%q', 'a string with "quotes" and \n newl ine')`
	
	produces the string:
	
	`"a string with \"quotes\" and \ newline"`
	
	The options `c`, `d`, `E`, `e`, `f`, `g`, `G`, `i`, `o`, `u`, `X`, and `x` all expect a number as argument. `q` and `s` expect a string. This function does not accept string values containing embedded zeros, except as arguments to the `q` option.

`string.len(s)`	Receives a string and returns its length. The empty string `""` has length `0`. Embedded zeros are counted, so `"a\000bc\000"` has length `5`.
`string.lower(s)`	Receives a string and returns a copy of this string with all uppercase letters changed to lowercase. All other characters are left unchanged.
`string.rep(s, n)`	Returns a string that is the concatenation of *n* copies of the string *s*.
`string.sub(s, i)` `string.sub(s, i, j)`	Returns the substring of *s* that starts at *i* and continues until *j*; *i* and *j* can be negative. If *j* is absent, it is assumed to be equal to `-1` (which is the same as the string length). In particular, the call `string.sub(s, 1, j)` returns a prefix of *s* with length *j*, and `string.sub(s, -i)` returns a suffix of *s* with length *i*.
`string.upper(s)`	Receives a string and returns a copy of this string with all lowercase letters changed to uppercase. All other characters are left unchanged.

Math library functions

This library is an interface to most of the functions of the ANSI C math library. All trigonometric functions work in radians. The functions `math.deg()` and `math.rad()` convert between radians and degrees. The math library functions are listed in the following table.

Function	Description
`math.abs(x)`	Returns the absolute value of *x*.
`math.acos(x)`	Returns the arc cosine of *x*.
`math.asin(x)`	Returns the arc sine of *x*.
`math.atan(x)`	Returns the arc tangent of *x*.

Function	Description
`math.atan2(y, x)`	Returns the arc tangent of y/x but uses the signs of both parameters to find the quadrant of the result (it also correctly handles the case of x being zero).
`math.ceil(x)`	Returns the smallest integer larger than or equal to x.
`math.cos(x)`	Returns the cosine of x.
`math.deg(x)`	Returns the angle x (given in radians) in degrees.
`math.exp(x)`	Returns the value e^x.
`math.floor(x)`	Returns the largest integer smaller than or equal to x.
`math.frexp(x)`	Returns `m` and `e` such that $x = m2^e$, where `e` is an integer and the absolute value of `m` is in the range `[0.5, 1]` (or zero when x is zero).
`math.ldexp(m, e)`	Returns $m2^e$ (`e` should be an integer).
`math.log(x)`	Returns the natural logarithm of x.
`math.log10(x)`	Returns the base-10 logarithm of x.
`math.max(x, ...)`	Returns the maximum value among its arguments.
`math.min(x, ...)`	Returns the minimum value among its arguments.
`math.pi`	The value of π (3.141592654).
`math.pow(x, y)`	Returns x^y (you can also use the expression `x^y` to compute this value).
`math.rad(x)`	Returns the angle x (given in degrees) in radians.

`math.random()`
`math.random(m)`
`math.random(m, n)`
	This function is an interface to the simple pseudorandom generator function `rand` provided by ANSI C.
	
	When called without arguments, returns a uniform pseudorandom real number in the range `[0,1]`. When called with an integer m, `math.random()` returns a uniform pseudorandom integer in the range `[1, m]`. When called with two integers m and n, `math.random()` returns a uniform pseudorandom integer in the range `[m, n]`.

`math.randomseed(x)`	Sets x as the seed for the pseudorandom generator; equal seeds produce equal sequences of numbers.
`math.sin(x)`	Returns the sine of x.
`math.sqrt(x)`	Returns the square root of x. You can also use the expression `x^0.5` to compute this value.
`math.tan(x)`	Returns the tangent of x.

Memory considerations for the runtime environment

The MP5103 reserves a large amount of memory for use with interactions with commands and test scripts. In addition, the installed modules reserve 512 MB for module-specific tasks, such as executing trigger models and processing reading buffers. The amount of memory usage is affected by the following product features:

- Reading buffers (including local default and user-created reading buffers; if they are on a remote node, they only affect the remote node); refer to Setting reading buffer capacity for additional information.
- Scripts that are loaded onto the instrument.
- Configuration lists.
- Scripts that are actively running.
- Variables that reside in the run-time environment.
- USB drives that contain files that use a lot of memory, even if the files are not related to instrument operation.

The more a feature is used or the larger its definition, the more memory it consumes. For normal usage, reading buffers commonly reserve large amounts of memory. The amount of memory used depends on the number of readings and the buffer style.

You can use the `slot[Z].bank[X].meminfo()` command to check the amount of module memory used for configuration lists and reading buffers as well as the total amount of memory allocated and used.

CAUTION: The MP5103 notifies you when the system runs out of memory. If the instrument encounters memory allocation errors (errors that specifically state “Out of Memory”), the state of the instrument cannot be guaranteed. After attempting to save any important data, turn off power to the instrument and turn it back on to reset the runtime environment and return the instrument to a known state. Unsaved scripts and reading buffers will be lost.

If you encounter memory problems, examine the test script or commands that were being executed when the memory problems occurred. Take action to reduce the size of the elements that are consuming memory. Also, consider using the `collectgarbage()` command to clean up unused memory. For information on `collectgarbage()`, refer to Base library functions (on page 38).

The buffer style is set when you create a reading buffer. The buffer style cannot be changed after the buffer has been created, so to eliminate memory problems caused by the style, you may need to delete or adjust the capacity of the buffers.

The amount of memory used by a sweep configuration is based on the number of source points. The actual memory consumption can vary greatly depending on how often the instrument takes readings during the sweep, but as a general rule, each source point can be expected to consume at least 24 bytes.

Bit manipulation and logic operations

The bit functions perform bitwise logic operations on two given numbers, and bit operations on one given number. Logic and bit operations truncate the fractional part of given numbers to make them integers.

Logic operations

The `bit.bitand()`, `bit.bitor()`, and `bit.bitxor()` functions in this group perform bitwise logic operations on two numbers. The Test Script Processor (TSP™) scripting engine performs the indicated logic operation on the binary equivalents of the two integers. This bitwise logic operation is performed on all corresponding bits of the two numbers. The result of a logic operation is returned as an integer.

Bit operations

The rest of the functions in this group are used for operations on the bits of a given number. You can use these functions to:

- Clear a bit
- Toggle a bit
- Test a bit
- Set a bit or bit field
- Retrieve the weighted value of a bit or field value

All these functions use an index parameter to specify the bit position of the given number. The least significant bit of a given number has an index of 1, and the most significant bit has an index of 32.

NOTE: The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

bit.bitand() (on page 129)
bit.bitor() (on page 130)
bit.bitxor() (on page 130)
bit.clear() (on page 131)
bit.get() (on page 132)
bit.getfield() (on page 133)
bit.set() (on page 134)
bit.setfield() (on page 135)
bit.test() (on page 136)
bit.toggle() (on page 137)

Data queue

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link™ network at any time

The data queue in the Test Script Processor (TSP™) scripting engine is first-in, first-out (FIFO).

You can access data from the data queue even if a remote group or a node has overlapped operations in process.

dataqueue.add() (on page 138)
dataqueue.CAPACITY (on page 139)
dataqueue.clear() (on page 139)
dataqueue.count (on page 140)
dataqueue.next() (on page 141)

File I/O

You can use the file I/O commands to open and close directories and files, write data, or to read a file on an installed USB flash drive. File I/O commands are organized into two groups:

- Commands that reside in the `fs` and `io` table, for example: `io.open()`, `io.close()`, `io.input()`, and `io.output()`. Use these commands to manage file system directories; open and close file descriptors; and perform basic I/O operations on a pair of default files (one input and one output).
- Commands that reside in the file descriptors (for example: *`fileVar`*`:seek()`, *`fileVar`*`:write()`, and *`fileVar`*`:read()`) operate exclusively on the file with which they are associated.

The root folder of the USB flash drive has the absolute path:

```
"/usb1/"
```

NOTE: You can use either the slash (`/`) or backslash (`\`) as a directory separator. However, the backslash is also used as an escape character, so if you use it as a directory separator, you generally need to use a double backslash (`\\`) when you are creating scripts or sending commands to the instrument.

Userstrings

Use the `userstring` functions to store and retrieve user-defined strings in nonvolatile memory. These strings are stored as key-value pairs. The key is a unique identifier such as a part number or identification string.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

TSP scripting

NOTE: You only need to review this information if you are using scripting and programming. Though it can improve your process to use scripts, you do not need to create scripts to use the instrument. Most of the examples in the documentation can be run by sending individual command messages.

A script is a collection of instrument control commands and programming statements. Scripting helps you combine commands into a block of code that the instrument can run.

Scripts offer several advantages compared to sending individual commands:

- Scripts are easier to save, refine, and implement than individual commands.
- The instrument process scripts more quickly and efficiently than individual commands.
- You can incorporate features such as looping and branching into scripts.
- Scripts allow the controller to perform other tasks while the instrument is running a script, enabling some parallel operation.
- Scripts eliminate repeated data transfer times from the controller.

In the instrument, the Test Script Processor (TSP™) scripting engine processes and runs scripts. Scripts can include combinations of Test Script Processor (TSP™) commands and Lua code.

Tools for managing scripts

You can use any of the following tools to manage scripts:

- The USB flash drive.
- Messages sent to the instrument.
- The Tektronix TSP Toolkit Visual Studio Code Extension, which is available at tek.com.
- Your own development tool or program.

NOTE: If you are using TSP Toolkit to create scripts, you do not need to use the commands `loadscript` and `endscript`.

Runtime and nonvolatile memory storage of scripts

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

The runtime environment is a collection of global variables, which include scripts, that the user has defined. A global variable can be used to store a value while the instrument is turned on. When you create a script, the instrument creates a global variable with the same name so that you can reference the script more conveniently. After scripts are loaded into the runtime environment, you can run and manage them from the front panel of the instrument or from a computer. Information in the runtime environment is lost when the instrument is turned off.

Nonvolatile memory is where information is stored even when the instrument is turned off. Save scripts to nonvolatile memory to save them through a power cycle or reset. The scripts that are in nonvolatile memory are loaded into the runtime environment when the instrument is turned on.

Scripts are placed in the runtime environment at the following times:

- When they are run.
- When they are loaded over a remote command interface.
- When the instrument is turned on (if they are stored in nonvolatile memory).

For detail on the amount of available memory, see Memory considerations for the runtime environment (on page 42).

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 Save a script through a power cycle (on page 47).

Script rules

You can have as many scripts as needed in the instrument. The only limitation is the amount of memory available to the runtime environment and the nonvolatile memory.

When a script is loaded into the runtime environment, a global variable with the same name as the script is created to reference the script.

Important points regarding scripts:

- Each script must have a unique name.
- Script names must not contain spaces.
- If you revise a script and save it to the instrument with a new name, the previously loaded script remains in the instrument with the original name.
- You can save scripts to nonvolatile memory in the instrument. Saving a script to nonvolatile memory allows the instrument to be turned off without losing the script.

Load a script

To load a script, you can:

- Copy it from a USB flash drive
- Use the `loadscript` and `endscript` commands over a remote command interface
- Use TSP Toolkit
- Create the script directly with commands
- Restore it from nonvolatile memory

Once a script is loaded into the instrument, you can execute it remotely.

You can have as many scripts in the instrument as can be supported by the memory available to the runtime environment.

Copy a script from a USB flash drive

You can copy a script from a USB flash drive. The script must contain the `loadscript` and `endscript` keywords with the `script.load()` command.

Refer to script.load() (on page 227) for detail.

Load a script with loadscript

To load a script over a remote interface, you can use the `loadscript` and `endscript` commands. Normally, when the instrument receives a command, it runs the command immediately. When the instrument receives the `loadscript` command, the instrument starts collecting subsequent messages instead of running them immediately.

The `endscript` command tells the instrument to stop collecting messages. It then compiles the collection of messages into a script that is stored as a function. This script is loaded into the runtime environment. When a script is loaded into the runtime environment with the `loadscript` commands, a global variable with the same name is created to reference the script.

NOTE: To store the script in nonvolatile memory, you need to use the *scriptVar*.save() command to save it in the instrument. For more information, refer to scriptVar.save() (on page 232).

Key points regarding scripts:

- If you load a new script with the same name as an existing script, the existing script is overwritten.
- Sending revised scripts with different names does not remove previously loaded scripts.
- You can save scripts to internal nonvolatile memory, which allows the instrument to be turned off or reset without losing the script.

To load a script using loadscript:

1. Send the command `loadscript` *scriptName*, where *scriptName* is the name of the script. The name must be a valid Lua variable name.
2. Send the commands that need to be included in the script.
3. Send the command `endscript`.
4. You can now run the script. Send the script name followed by `()`.

The following example creates and runs a script named `InstrumentInfo` that returns the serial numbers of the mainframe and any modules installed in the instrument. If a module is not installed, an error is displayed.

```
loadscript InstrumentInfo
   print(localnode.serialno)
   print(slot[1].serialno)
   print(slot[2].serialno)
   print(slot[3].serialno)
endscript
InstrumentInfo()
```

Load a script using TSP Toolkit

If you use TSP Toolkit to load a script, you do not need to include the `loadscript` and `endscript` commands.

NOTE: For more information on TSP Toolkit, visit the Tektronix TSP Toolkit extension page in the Microsoft Visual Studio Code Marketplace.

To load a script using TSP Toolkit:

1. Open Microsoft Visual Studio Code.
2. Select the Explorer and select **New File**.
3. Assign a file name. Include the file extension `.tsp`.
4. Create the script.
5. Save the script.
6. Select the TSP Toolkit extension.
7. Connect to the mainframe.
8. Right-click the code and select **Send Script to Terminal**.

Create a script on the instrument with commands

To create a new script on the instrument using TSP commands, send:

```
scriptVar = script.new("code", "name")
```

For detail on using this command, refer to script.new() (on page 228).

Restore a script from nonvolatile memory

If a script has been saved to nonvolatile memory, it can be loaded into the runtime environment by sending:

```
script.restore("name")
```

Refer to script.restore() (on page 228) for detail.

Save a script through a power cycle

You can save scripts to nonvolatile memory. To keep a script through a power cycle, you must save the script to nonvolatile memory.

To save a script to nonvolatile memory:

1. Create and load a script.
2. Send the command *scriptVar*.save(), where *scriptVar* is the name of the script.

Save a script to a USB flash drive

To save a script to an external USB flash drive:

1. Load a script.
2. Insert a USB flash drive into the USB port.
3. Send the command *scriptVar*.save("/usb1/filename.tsp"), where *scriptVar* is the variable referencing the script and filename is the name of the file.

For more detail, refer to scriptVar.save() (on page 232).

Run a script

You can run any script using *scriptVar*.run(). Replace *scriptVar* with the name of a script that is in nonvolatile or runtime memory.

Rename a script

To rename a script in the runtime environment:

1. Load the script into the runtime environment with a different name.
2. Delete the previous version of the script.

To rename a script in nonvolatile memory:

Send the commands:

```
script.restore("oldName")
newName = oldName
newName.name = "newName"
newName.save()
script.delete("oldName")
```

This example restores the script to be renamed to the runtime environment, assigns it to a new variable with the new name, renames the script, and saves it to memory again before the old named script is deleted.

Delete a script

To delete a script from nonvolatile memory, send the command

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

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

To delete a script from runtime memory, you can do any of the following:

- Set the name of the script to nil and call `collectgarbage()`
- Reboot the instrument, which clears all variables in the runtime environment

View scripts

To retrieve a list of scripts in nonvolatile memory, use `script.table()`.

Refer to script.table() (on page 229) for additional information.

Retrieve the content of a script

To retrieve the content of a script, use *scriptVar*`.source`, where *scriptVar* is the name of the script you want to retrieve. For example, to retrieve a script named `contactTest`, send:

```
print(contactTest.source)
```

The source code is returned as a single string. The `loadscript` and `endscript` keywords are not included.

To retrieve the content of a script with the `loadscript` and `endscript` keywords included, use *scriptVar*`.list()`. For example:

```
contactTest.list()
```

For more detail, refer to scriptVar.source (on page 232) and scriptVar.list() (on page 229).

Commands that cannot be used in scripts

You cannot use the following commands as variables in scripts:

NOTE: Some functions resemble the strings in the following tables but are actually defined TSP functions. For example, printbuffer() (on page 222) is a function that you can use in scripts. If you are uncertain, check the TSP command reference (on page 129) to verify that the string is part of a defined function.

- `abort`
- `bit`
- `endflash`
- `endscript`
- `flash`
- `fs`
- `io`
- `loadscript`
- `loadandrunscript`
- `login`
- `logout`
- `node`
- `opc`
- `password`
- `prevflash`
- `printbuffer`
- `printnumber`
- `restoreglobals`
- `table`
- `waitcomplete`

Common commands cannot be used in scripts. The following table lists equivalent TSP commands that can be used in scripts.

Unavailable commands with TSP equivalents

Common commands	TSP equivalent commands
*CLS	eventlog.clear() status.reset()
*ESE	status.standard.enable
*ESE?	print(status.standard.enable)
*ESR?	print(status.standard.event)
*IDN?	print(localnode.model) print(localnode.serialno) print(localnode.revision)
*OPC	opc()
*OPC?	waitcomplete() print([[1]])
*RST	reset()
*SRE	status.request_enable
*SRE?	print(status.request_enable)
*STB?	print(status.condition)
*TRG	No equivalent
*TST?	print([[0]])
*WAI	waitcomplete()

Configuration lists and sweeps

Configuration lists

A configuration is a table of stored source and measure settings for a specified channel. A configuration list is made up of configurations, indicated by multiple indexes.

You can recall a specific configuration index or you can sequence through a configuration list using the trigger model.

The first time you store a configuration index, the instrument stores the settings in configuration index 1. Subsequent indexes are numbered sequentially. You can use the index number to identify a specific configuration. The number of configuration lists and number of indexes in the configuration lists that can be saved depends on the available memory.

The settings that are stored in configuration list indexes are listed in Settings stored in a configuration index (on page 55).

Work with configurations

Source and measure settings for a channel can be stored in a set of tables called a configuration. A configuration can used to save settings and restore them to a channel.

Not all settings are saved to a configuration when one is created. Once a configuration is saved, it may be modified before either being stored in a configuration list or recalled to a channel. If an incomplete configuration is restored, the settings that do not exist are not changed from their active states.

Save the default settings to a configuration

You can save a set up that contains the default values for the channel settings regardless of present settings using the slot[Z].psu[X].config.default() (on page 369) or slot[Z].smu[X].config.default() (on page 474) command.

To save default settings of a PSU, send:

```
newDefaultPSUConfig = slot[2].psu[1].config.default()
```

This example saves the present default channel settings of channel 1 of the PSU in slot 2 to `newDefaultPSUConfig`.

To save default settings of a SMU, send:

```
newDefaultSMUConfig = slot[1].smu[1].config.default()
```

This example saves the present default channel settings of channel 1 of the SMU in slot 1 to `newDefaultSMUConfig`.

Save the active settings to a configuration

You can save settings that are presently configured to a configuration using the slot[Z].psu[X].config.active() (on page 369) or slot[Z].smu[X].config.active() (on page 473) command. This allows you to create custom setups for an instrument channel that can be restored.

For a PSU, send:

```
newActivePSUConfig = slot[2].psu[1].config.active()
```

This example saves the configuration table of channel 1 of the PSU installed in slot 2 to `newActivePSUConfig`.

For an SMU, send:

```
newActiveSMUConfig = slot[1].smu[1].config.active()
```

This example saves the configuration table of channel 1 of the SMU installed in slot 1 to `newActiveSMUConfig`.

Apply a configuration to a channel

If you recall a configuration, the active settings change to match the configuration. If a setting is not defined, the previous setting is preserved. You can apply configurations using the slot[Z].psu[X].config.set() (on page 370) or slot[Z].smu[X].config.set() (on page 474) command.

For a PSU, send:

```
slot[2].psu[1].config.set(newActivePSUConfig)
```

This example applies the configuration `newActivePSUConfig` to channel 1 of the PSU installed in slot 2.

For an SMU, send:

```
slot[1].smu[1].config.set(newActiveSMUConfig)
```

This example applies the configuration `newActiveSMUConfig` to channel 1 of the SMU installed in slot 1.

Work with configuration lists and indexes

A configuration list consists of a reference configuration and a number of configurations stored at indexes. The reference configuration can be useful to track changes in the configurations at each step.

To store an instrument configuration in a configuration list, you need to:

- Set up the reference configuration
- Create a configuration list
- Configure the channel with the settings
- Store the settings into a configuration index in the specified configuration list

After you store configuration indexes to a configuration list, you can do the following operations:

- Recall a configuration index to restore the settings to the channel
- View the contents of a configuration index, including the reference configuration
- Delete a configuration index
- Delete the entire configuration list

You work with configuration lists by using remote commands.

Create a configuration list

NOTE: Configuration list names must be unique to a slot, but configuration lists can have the same name on different slots. For example, if `ListA` was created for channel 1 on slot 1, another list cannot be named `ListA` for channel 2 on slot 1. However, you may create `ListA` for a channel on slot 2.

You can create a configuration list with the slot[Z].psu[X].configlist.create() (on page 372) or slot[Z].smu[X].configlist.create() (on page 476) command.

When the configuration list is created, a configuration may be specified as the reference configuration. If no configuration is provided in the command, the present setup is used.

The reference configuration is used to fill in missing settings when configurations are added to the configuration list. For example, if the reference configuration lists source level as 5 V and a configuration is saved that does not have a source level specified, the level is set to 5 V in that index.

To create a configuration list on a PSU:

```
slot[2].psu[1].configlist.create("MyConfigList")
```

This example creates a configuration list named `MyConfigList` using the present setup on channel 1 of the PSU in slot 2 as the reference.

To create a configuration list on a SMU:

```
slot[1].smu[1].configlist.create("MyConfigList")
```

This example creates a configuration list named `MyConfigList` using the current setup on channel 1 of the SMU in slot 1 as the reference.

Store settings into a configuration list index

A configuration index contains a copy of the instrument source and measure settings at a specific time.

To add configurations to a configuration list, use the following commands:

- To add a configuration to the end of a list, use slot[Z].psu[X].configlist.append() (on page 371) or slot[Z].smu[X].configlist.append() (on page 475).
- To overwrite an existing index with a new configuration, use slot[Z].psu[X].configlist.store() (on page 378) or slot[Z].smu[X].configlist.store() (on page 482).

If no existing configuration is provided with these commands, the present settings are saved.

The following examples demonstrate how to store channel settings to a configuration list.

Store settings for a channel

The following example creates a PSU configuration list and indexes and does the following actions:

- Creates a configuration list named `MyConfigList` on the PSU module in slot 2.
- Sets the source level for channel 1 to 2 V.
- Sets the measure rate for channel 1 to fast.
- Stores the settings to index 1 of `MyConfigList`.
- Changes the source level for channel 1 to 3 V.
- Changes the measure rate for channel 1 to normal.
- Stores the settings to index 2 of `MyConfigList`.

Send the following TSP commands:

```
slot[2].psu[1].configlist.create("MyConfigList")
slot[2].psu[1].source.levelv = 2
slot[2].psu[1].measure.rate = slot[2].psu[1].RATE_FAST
slot[2].psu[1].configlist.append("MyConfigList")
slot[2].psu[1].source.levelv = 3
slot[2].psu[1].measure.rate = slot[2].psu[1].RATE_NORMAL
slot[2].psu[1].configlist.append("MyConfigList")
```

The following example creates a SMU configuration list and indexes and does the following actions:

- Creates a configuration list on the SMU module in slot 1 named `MyConfigList`.
- Sets the source function to voltage.
- Sets the source range to 20 V.
- Sets the source level to 2 V.
- Sets the current measure range to 100 mA.
- Stores the settings to index 1 of `MyConfigList`.
- Changes the source level to 4 V.
- Changes the measure range to 10 mA
- Stores the settings to index 2 of `MyConfigList`.

Send the following TSP commands:

```
slot[1].smu[1].configlist.create("MyConfigList")
slot[1].smu[1].source.func = slot[1].smu[1].FUNC_DC_VOLTAGE
slot[1].smu[1].source.rangev = 20
slot[1].smu[1].source.levelv = 2
slot[1].smu[1].measure.rangei = 100e-3
slot[1].smu[1].configlist.append("MyConfigList")
slot[1].smu[1].source.levelv = 4
slot[1].smu[1].measure.rangei = 10e-3
slot[1].smu[1].configlist.append("MyConfigList")
```

Recall a configuration

You can recall the configuration stored in a specific index in a configuration list and apply those settings to the channel using configuration list commands or within a trigger model. The configuration list-specific commands are slot[Z].psu[X].configlist.recall() (on page 376) and slot[Z].smu[X].configlist.recall() (on page 480).

If you recall an index outside of a trigger model, you must specify an index to recall. For example, to recall index 2 from a configuration list on a SMU, send:

```
slot[1].smu[1].configlist.recall("MyConfigList", 2)
```

In a trigger model, you can recall the next, previous, or a specific index.

To recall the next index in a configuration list within a trigger model, use the slot[Z].trigger.model.addblock.configlist.next() (on page 580) command. If no index has been called, this command recalls index 1.

To recall the previous index in a configuration list in a trigger model, use the slot[Z].trigger.model.addblock.configlist.prev() (on page 582) command. If no index was previously called, it recalls the final index.

To recall a specific index in a trigger model, use the slot[Z].trigger.model.addblock.configlist.recall() (on page 585) command.

These blocks can be combined with branch counter blocks to loop through configuration indexes. See Trigger models (on page 69) for more information.

Query the configuration in an index

You can return the configuration table of settings saved as a specified index using the slot[Z].psu[X].configlist.query() (on page 375) or slot[Z].smu[X].configlist.query() (on page 479) commands. If the optional boolean parameter `diff` is not specified or is `false`, the table returned contains all settings. If `diff` is `true`, only the settings that differ from the reference configuration are returned in a Lua table.

The returned configuration table can be accessed using the same command structure as TSP commands. See Settings stored in a configuration index (on page 55) for more detail.

This command does not apply the settings to the channel.

The following example demonstrates creating a configuration list on a PSU with multiple indexes, querying one index, and returning the index value:

```
slot[1].psu[1].configlist.delete("MyConfigList")
slot[1].psu[1].configlist.create("MyConfigList")
slot[1].psu[1].source.levelv= 0.2
slot[1].psu[1].configlist.store("MyConfigList", 1)
slot[1].psu[1].source.levelv= 0.3
slot[1].psu[1].measure.rel.levelv = slot[1].psu[1].measure.v()
slot[1].psu[1].configlist.store("MyConfigList", 2)
slot[1].psu[1].source.levelv= 0.5
slot[1].psu[1].configlist.store("MyConfigList",3)
config2 = slot[1].psu[1].configlist.query("MyConfigList", 2)
print(config2.source.levelv)
```

This example ensures that any existing configuration lists named `MyConfigList` are deleted, then creates a new configuration list named `MyConfigList`. The source level is set to 0.2 V, then the configuration is stored at index 1 of `MyConfigList`. Then the source level is set to 0.3 V, a measurement is made to establish the relative offset, and the relative offset is set to the measured value. The changes are saved to index 2 of `MyConfigList`. The source level is then set to 0.5 V and the configuration is stored at index 3 of `MyConfigList`. Finally, `MyConfigList` is queried at index 2. The value 0.3 V is returned.

Return the reference configuration

You can return the configuration table of settings that is used as the reference for a configuration list using the slot[Z].psu[X].configlist.reference() (on page 377) or slot[Z].smu[X].configlist.reference() (on page 481) command.

This command does not apply the settings to the channel.

To return the reference configuration on a PSU:

```
ref = slot[1].psu[1].configlist.reference("MyConfigList")
```

This example returns the reference configuration table for `MyConfigList`.

To return the reference configuration on a SMU:

```
ref = slot[1].smu[1].configlist.reference("MyConfigList")
```

This example returns the reference configuration table for `MyConfigList`.

Delete a configuration index

You can delete a single configuration index from a configuration list using the slot[Z].psu[X].configlist.delete() (on page 373) or slot[Z].smu[X].configlist.delete() (on page 477) command. The specified index is removed from the list and the following indexes are renumbered so that the indexes are numbered sequentially.

When deleting multiple indexes, start with the highest numbered index. To delete all indexes, delete index 1 until the configuration list is empty.

To delete an index on a PSU:

```
slot[1].psu[1].configlist.delete("MyConfigList", 2)
```

This example deletes index 2 from a configuration list named `MyConfigList` on PSU slot 1 channel 1.

To delete an index on a SMU:

```
slot[1].smu[1].configlist.delete("MyConfigList", 2)
```

This example deletes index 2 from a configuration list named `MyConfigList` on SMU slot 1 channel 1.

Delete a configuration list

You can delete an entire configuration list using the slot[Z].psu[X].configlist.deletelist() (on page 374) or slot[Z].smu[X].configlist.deletelist() (on page 478) command. This removes all indexes and stored configurations and deletes the list name.

To delete a configuration list on a PSU:

```
slot[1].psu[1].configlist.deletelist("MyConfigList")
```

This example deletes the configuration list `MyConfigList` from PSU channel 1 in slot 1.

To delete a configuration list on a SMU:

```
slot[2].smu[1].configlist.deletelist("MyConfigList")
```

This example deletes the configuration list `MyConfigList` from SMU channel 1 in slot 2.

List the available configuration lists

You can view the names of the configuration lists using the slot[Z].psu[X].configlist.table() (on page 379) or slot[Z].smu[X].configlist.table() (on page 483) command. The names are returned as a Lua table.

To list available configuration lists on a SMU:

```
myConfigLists = slot[1].smu[1].configlist.table()
for i, name in pairs(myConfigLists) do
 print(name)
end
```

This example prints the names of all configurations on channel 1 of the SMU installed in slot 1.

Determine the number of indexes in a configuration list

You can return the number of indexes in a specified configuration list. Use the slot[Z].psu[X].configlist.size() (on page 377) or slot[Z].smu[X].configlist.size() (on page 481) command.

To view the number of indexes on a SMU:

```
print(slot[1].smu[1].configlist.size("MyConfigList"))
```

This example prints the number of indexes saved in `MyConfigList` on channel 1 of the SMU in slot 1.

Settings stored in a configuration

The settings stored in a configuration are accessed using the same structure as the corresponding TSP attribute command for that setting. The saved configuration in the table access commands is represented with *`configVar`*.

The following settings are stored in a configuration index for PSU modules.

Source settings	Description	Table access command
	Level	*configVar*.source.levelv
	Limit level	*configVar*.source.limiti
	Output-off mode	*configVar*.source.offmode
	Overcurrent protection enable	*configVar*.source.protect.enablei
	Overvoltage protection enable	*configVar*.source.protect.enablev
	Overcurrent protection level	*configVar*.source.protect.leveli
	Overvoltage protection level	*configVar*.source.protect.levelv
	Slew rate, falling	*configVar*.source.slewrate.fallv
	Slew rate, rising	*configVar*.source.slewrate.risev
	Slew rate, rising and falling	*configVar*.source.slewrate.v

Measure settings	Description	Table access command
	Count Measure rate Relative offset enable Relative offset level Temperature compensation Overlapped mode	*configVar*.measure.count *configVar*.measure.rate *configVar*.measure.rel.enableY *configVar*.measure.rel.levelY *configVar*.measure.tempcomp *configVar*.overlapped

The following settings are stored in a configuration index for SMU modules:

Source settings	Description	Table access command
	Autorange Delay Function Level Negative limit level Positive limit level Symmetrical limit level Autorange low limit Output off function Output off negative limit level Output off positive limit level Output off symmetrical limit level Output off mode Range	*configVar*.source.autorangeY *configVar*.source.delay *configVar*.source.func *configVar*.source.levelY *configVar*.source.limitnY *configVar*.source.limitpY *configVar*.source.limitY *configVar*.source.lowrangeY *configVar*.source.offfunc *configVar*.source.offlimitnY *configVar*.source.offlimitpY *configVar*.source.offlimitY *configVar*.source.offmode *configVar*.source.rangeY

Measure settings	Description	Table access command
	Aperture	*configVar*.measure.aperture
	Autorange	*configVar*.measure.autorangeY
	Contact check speed	*configVar*.measure.contact.speed
	Contact check threshold	*configVar*.measure.contact.threshold
	Count	*configVar*.measure.count
	Delay	*configVar*.measure.delay
	Interval	*configVar*.measure.interval
	Autorange low limit	*configVar*.measure.lowrangeY
	NPLC	*configVar*.measure.nplc
	Range	*configVar*.measure.rangeY
	Relative offset enable	*configVar*.measure.rel.enableY
	Relative offset level	*configVar*.measure.rel.levelY
	Overlapped mode	*configVar*.overlapped
	Sense mode	*configVar*.sense

Sweep operation

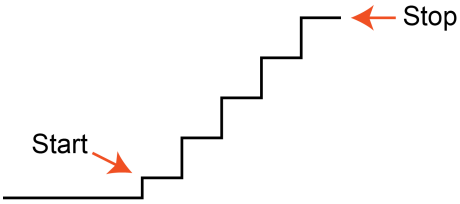
The MP5103 can generate DC and pulse sweeps using the trigger model and the sweep configuration commands.

The sweep configuration commands generate a sweep table for a specified channel that can then be iterated through in the trigger model. Sweeps can be DC, where the source level steps each time, or the sweep can be pulsed, where the source level returns to a bias value between making steps. For either, the sweep table is the same, but the trigger model configuration changes.

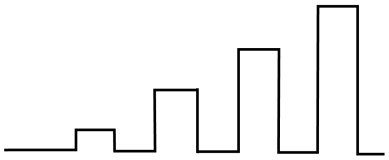
The following sections provide an overview of sweep options and how to configure them for the MP5000.

Linear staircase sweeps

A linear staircase sweep increases or decreases the source level in fixed steps from the start value to the stop value. The following figures show an increasing linear staircase sweep and a pulsed staircase sweep. Pulsed linear staircase sweeps function the same way that DC linear staircase sweeps function, except that pulsed linear staircase sweeps return to the idle level between pulses.



DC linear staircase sweep



Pulsed linear staircase sweep

A linear sweep is configured using the slot[Z].psu[X].trigger.source.linearv() (on page 405) command or the slot[Z].smu[X].trigger.source.linearY() (on page 532) command, where `Y` is either `v` = voltage or `i` = current. The command parameters define the start and stop levels for the sweep and the number of points in the sweep. The step size is determined using the following formula:

step = (stop - start) / (points - 1)

Conversely, the number of points can be calculated if the step size is known:

points = 1 + (stop - start) / step

For example, to create the sweep table for a linear sweep from 0 to 10 V in 1 V steps on a PSU channel 1 in slot 1:

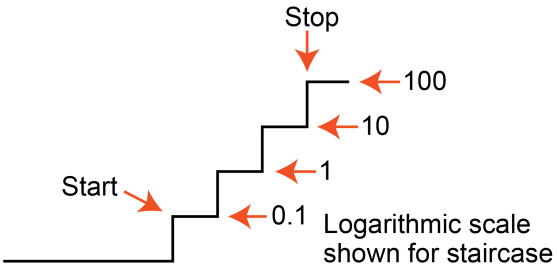
1. Calculate the number of points: 1 + (10 - 0)/1 = 11 points
2. Send the command

```
slot[1].psu[1].trigger.source.linearv(0, 10, 11)
```

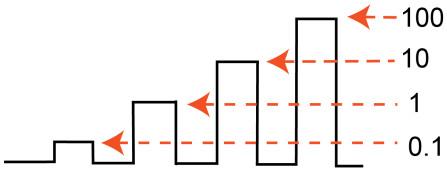
Logarithmic staircase sweeps

This type of sweep is similar to the linear staircase sweep. The steps, however, are done on a logarithmic scale, so the step size increases or decreases exponentially. You can also use an asymptote to control the inflection.

The sweep table for a log sweep is configured using the slot[Z].psu[X].trigger.source.logv() (on page 408) or slot[Z].smu[X].trigger.source.logY() (on page 535) command, where `Y` is either `v` = voltage or `i` = current.



DC logarithmic staircase sweep



Pulsed logarithmic staircase sweep

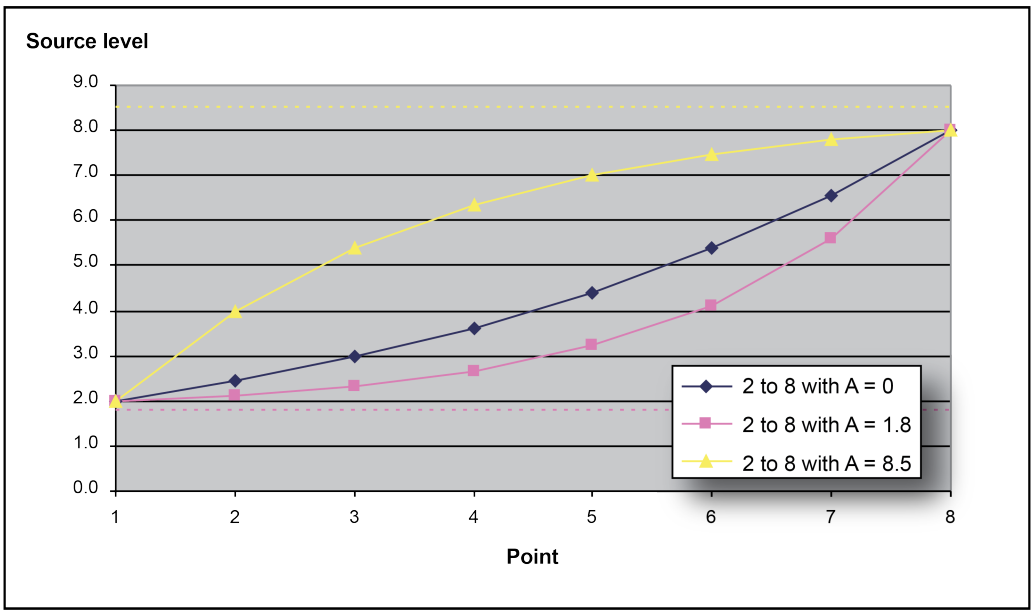
The formula for a logarithmic sweep is:

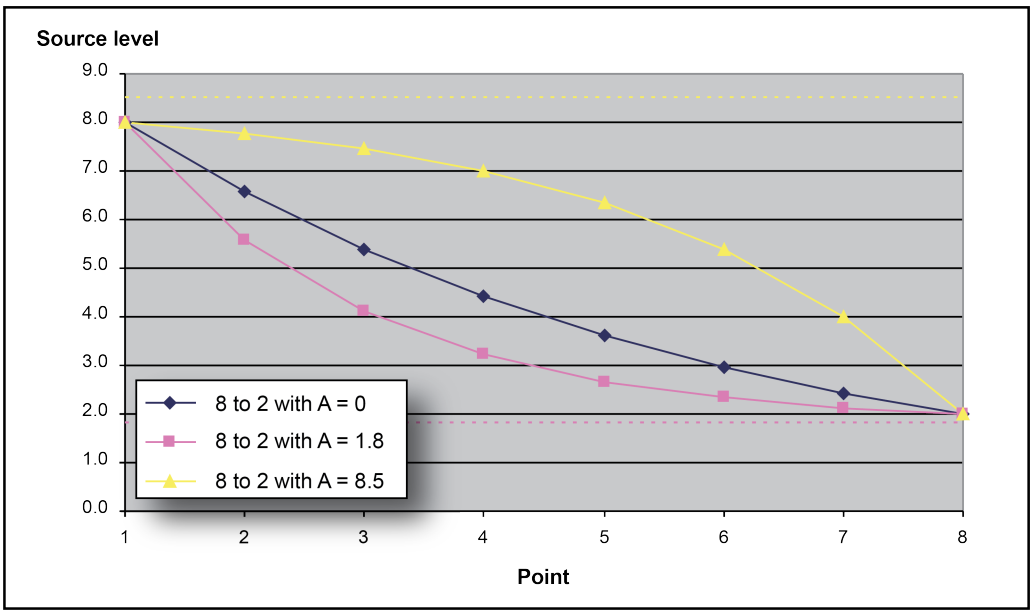
$v_i = A + kb^i$

Where:

v_i	=	The source value at source point *i*
i	=	The index of points in the sweep (ranges from 0 to *N* − 1), where *N* is the number of points in the sweep
k	=	The initial source value as an offset from the asymptote
b	=	The step size ratio
A	=	The asymptote value

The asymptote is used to change the inflection of the sweep curve and allow it to sweep through zero. The following figures depict the effect of the asymptote on the inflection of the sweep curve.





Solving for *k* and *b* provides the following formulas:

$$k = V_{start} - A$$

$$b = 10^{\left(\frac{\log10(V_{end} - A) - \log10(V_{start} - A)}{N - 1}\right)}$$

Where:

V_{end} = The source value at the end point

V_{start} = The source value at the start point

N = The number of points in the sweep

A = The asymptote value

NOTE: The number of points in a sweep is one greater than the number of steps in the sweep.

For example, to generate the sweep table for a logarithmic sweep from 1 V to 10 V in 5 steps on a SMU channel 1 in slot 2:

1. Calculate the number of points in the sweep. Since logarithmic scans are always one point greater than the number of steps, the total number of points is 6.
2. Send the command

```
slot[2].smu[1].trigger.source.logv(1, 10, 6)
```

We can calculate the steps in the sweep table using the following:

$A = 0$, $V_{start} = 1$, $V_{end} = 10$, $N = 5$

Using the formula above, $k = 1$

Step size (b) for the sweep in the above figure is calculated as follows:

$$\text{Log step size} = \frac{\log10(\text{stop}) - \log10(\text{start})}{\text{Points} - 1}$$

$$= \frac{\log10(10) - \log10(1)}{5 - 1}$$

$$= \frac{1 - 0}{4}$$

$$= 0.25$$

Therefore, $b = 10^{(\log \text{ step size})} = 1.7783$

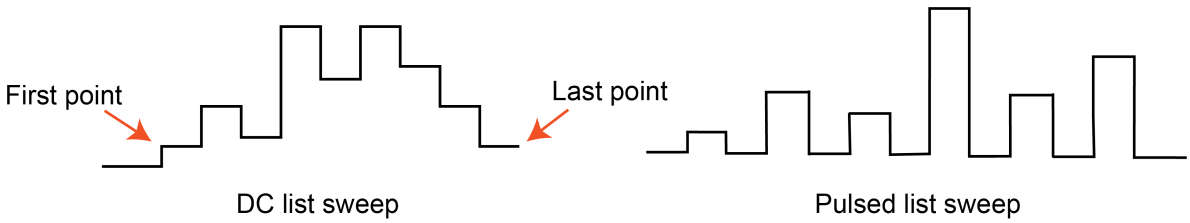
The log steps for this sweep are listed in the following table.

Logarithmic sweep points		
Source point *(N)*	**Source level *(V)***	**Step number *(i)***
1	1	0
2	1.7783	1
3	3.1623	2
4	5.6234	3
5	10	4

List sweeps

Use a list sweep to configure a sweep with arbitrary steps. When enabled, a measurement is made at each point after a source and measurement settling time. The sweep table for a list sweep is configured using the slot[Z].psu[X].trigger.source.listv() (on page 406) or slot[Z].smu[X].trigger.source.listY() (on page 533) function, where `Y` is either `v` = voltage or `i` = current. This function configures the source values that the PSU or SMU outputs when performing a list sweep.

The following figures show a list sweep with arbitrary steps and a pulsed list sweep. Pulsed list sweeps function the same way that DC list sweeps function, except that pulsed list sweeps return to the idle level between pulses.



DC list sweep Pulsed list sweep

To generate the sweep table for a voltage list sweep on channel 1 of a PSU module in slot 1:

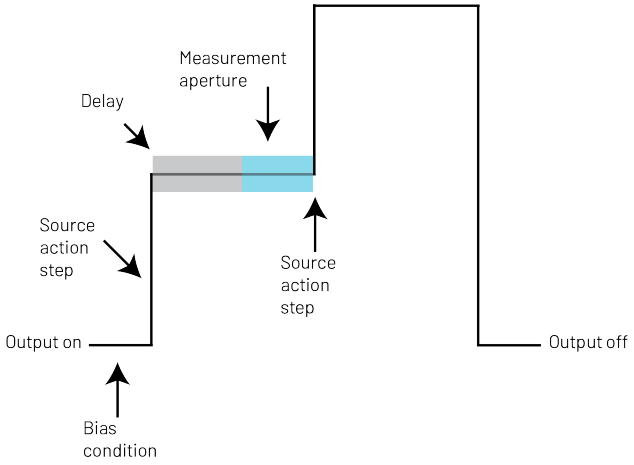
1. Determine the point sequence. For this example, the sweep contains the values 3 V, 1 V, 4 V, 5 V and 2 V.
2. Send the command:
   ```
   slot[1].psu[1].trigger.source.listv({3, 1, 4, 5, 2})
   ```

Run a DC sweep

Once a sweep table is created, it is executed using the trigger model. Specifically, the source action blocks are used to iterate through the sweep table. The blocks listed in the following table are available.

Name	Command	Description
Source action bias	`slot[Z].trigger.model.addblock.source.action.bias()`	Sets the output level to the level defined by the `*.source.levelY` command (PSU or SMU version)
Source action set	`slot[Z].trigger.model.addblock.source.action.set()`	Sets the output level to the value at the index without advancing the index
Source action skip	`slot[Z].trigger.model.addblock.source.action.skip()`	Moves to the next index without changing the source level
Source action step	`slot[Z].trigger.model.addblock.source.action.step()`	Moves to the next index and changes the source level to that value

The TriggerFlow model gives you the flexibility to control all the aspects of your sweep with additional blocks. Sweep timing is controlled using constant delay blocks or wait on event blocks to synchronize with events.



Sweep flow is controlled by branch blocks and timing blocks. Use branch counter blocks to simplify repetitive sweeping actions and iterate through sweep steps. Use other branching blocks to skip steps of a sweep as needed. The actual number of sweep steps that are executed depends on this flow and the number of times the source action step or source action skip blocks are executed.

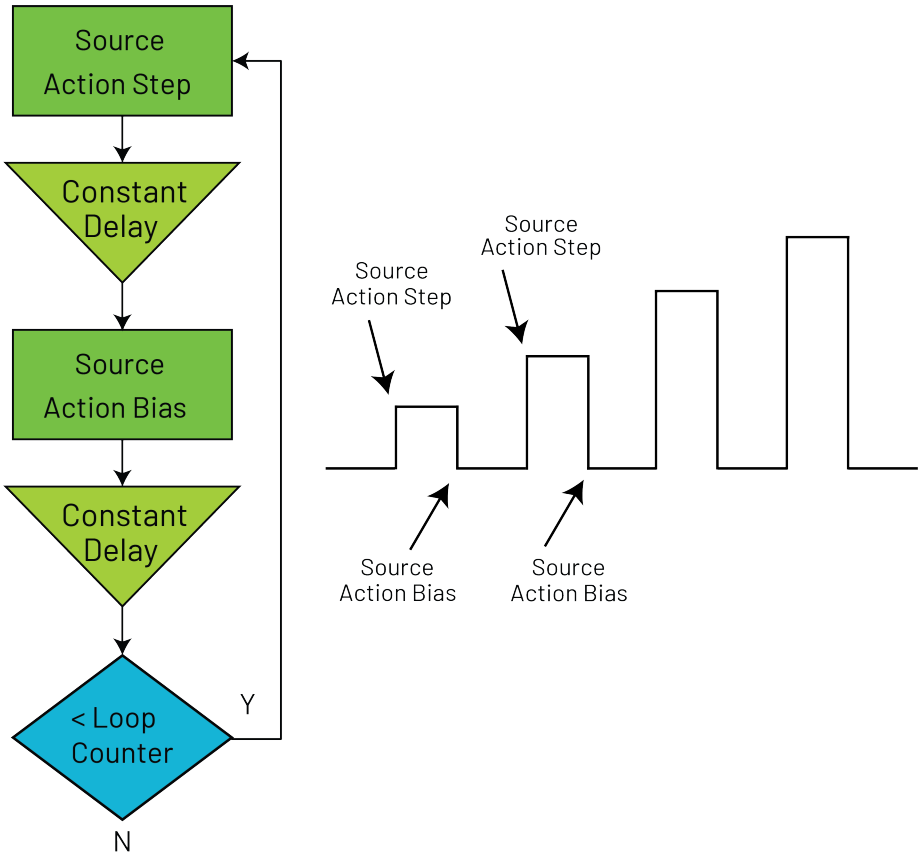
Measurements are made during the sweep using measurement blocks. If your module supports overlapped measurements, measurements can be made in the background during trigger model execution across the entire sweep. For more information on creating and building a trigger model, see Trigger Models (on page 69).

Run a pulse sweep

You can use the trigger model to convert a DC sweep that was configured using the sweep commands into a pulse sweep.

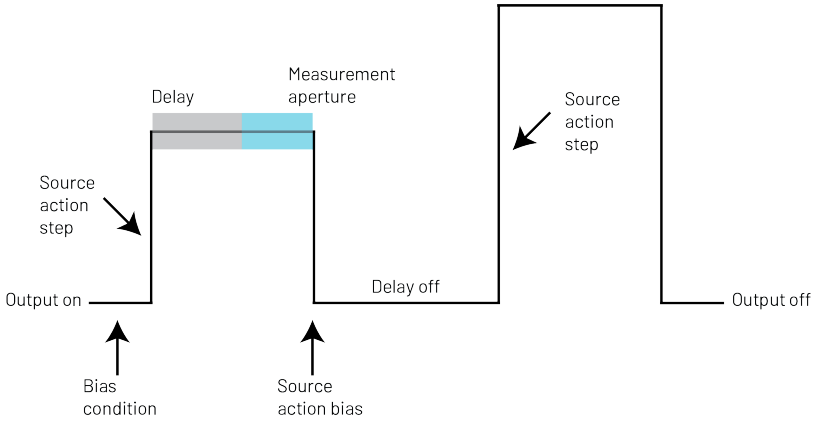
When building the trigger model for a pulsed sweep, use the source action bias block between each source action step block. The bias block returns the source level to the level set by the slot[Z].smu[X].source.levelY (on page 514) or slot[Z].psu[X].source.levelv (on page 392) commands. The next step block sets the source level to the next level in the sweep, resulting in a pulse effect.

Pulse timing, sweep flow, and measurements are all determined using other blocks in the trigger model. This is the same operation as DC sweeps. Pulse rise and fall times cannot be directly controlled using the trigger model. For PSU modules, the rate at which source values change affected by slew rate, load, and limit. For SMU modules, the rate at which source values change depends on load, source function, ranges, and limits. For more information on creating and building a trigger model, see Trigger models (on page 69).



Pulse sweeps using the extended operating area (EOA) of a supported SMU module may be operated in the same manner using the trigger model.

When not sourcing in the EOA, the source level and limit is set by the values established before the trigger model was started. This is called the bias condition. The bias condition is set with the `slot[Z].smu[X].source.levelY` (on page 514) and `slot[Z].smu[X].source.limitY` (on page 517) commands. The values must be compatible with the DC or continuous operating area of the SMU as well as with the selected fixed ranges.



Abort a sweep

You can abort a sweep by stopping the trigger model. See Abort the trigger model (on page 90) for more information.

Sweep programming examples

The following topics show examples of configured sweeps and trigger models that execute those sweeps.

Sweep type	Parameters for sweep examples
PSU linear sweep (on page 65)	PSU slot 2, channel 1 Linear sweep, 0 V to 10 V 200 ms delay on each step
SMU linear sweep with continuous measurement (on page 65)	SMU slot 1, channel 1 Linear sweep, 0 V to 10 V 200 ms delay on each step Overlapped measurement during sweep
PSU list sweep with timed measurement (on page 66)	PSU slot 2, channel 1 List sweep 200 ms delay on each step Single measure after each delay

Sweep type	Parameters for sweep examples
SMU pulse sweep (on page 66)	SMU slot 1, channel 1 Linear pulse sweep 0 V to 10 V 0 V bias 100 ms pulses
SMU pulse sweep (on page 66) (extended operating area)	

PSU linear sweep example

Parameters

- PSU slot 2, channel 1
- Linear sweep, 0 V to 10 V
- 200 ms delay on each step

Code

```
-- Alias PSU.
local psu = slot[2].psu[1]
-- Configure the linear sweep.
psu.trigger.source.linearv(0, 10, 11)
-- Alias the trigger model.
local triggerModel = slot[2].trigger.model
triggerModel.create("tm")
triggerModel.addblock.source.output("tm", "output-on", 1, 1)
-- Step to the next source voltage in the sweep.
triggerModel.addblock.source.action.step("tm", "step-voltage", 1)
-- Delay for 200 ms.
triggerModel.addblock.delay.constant("tm", "delay", 200e-3)
-- Loop to the voltage step.
triggerModel.addblock.branch.counter("tm", "branch", "step-voltage", 11)
triggerModel.addblock.source.output("tm", "output-off", 1, 0)
-- Initiate the trigger model.
triggerModel.initiate("tm")
```

SMU linear sweep with continuous measurement example

Parameters

- SMU slot 1, channel 1
- Linear sweep, 0 V to 10 V
- 200 ms delay on each step
- Overlapped measurement during sweep

Code

```
-- Alias the SMU.
local smu = slot[1].smu[1]
-- Configure the linear sweep.
smu.trigger.source.linearv(0,10,11)
smu.source.levelv = 0
smu.source.rangev = 10
-- Configure the measurement in the trigger model.
smu.defbuffer1.clear()
smu.defbuffer2.clear()
smu.trigger.measure.iv(smu.defbuffer1, smu.defbuffer2)
-- Alias the trigger model.
local triggerModel = slot[1].trigger.model
triggerModel.delete("tm")
```

```
triggerModel.create("tm")
triggerModel.addblock.source.output("tm", "output-on", 1, 1)
-- Start overlapped measurements.
triggerModel.addblock.measureoverlapped("tm", "measure", 1, 200)
-- Step to the next source voltage in the sweep.
triggerModel.addblock.source.action.step("tm", "step-voltage", 1)
-- Delay for 200 ms.
triggerModel.addblock.delay.constant("tm", "delay", 200e-3)
-- Loop to the voltage step.
triggerModel.addblock.branch.counter("tm", "branch", "step-voltage", 11)
triggerModel.addblock.source.output("tm", "output-off", 1, 0)
-- Initiate the trigger model.
triggerModel.initiate("tm")
```

PSU list sweep with timed measurement example

Parameters

- PSU slot 2, channel 1
- List sweep
- 200 ms delay on each step
- Single measure after each delay

Code

```
-- Alias the PSU.
local psu = slot[2].psu[1]
-- Configure the list sweep.
psu.trigger.source.listv({10,8,4,2,1,0,1,2,4,8,10})
-- Configure the buffer for measurements.
psu.trigger.measure.i(psu.defbuffer1)
-- Alias the trigger model.
local triggerModel = slot[2].trigger.model
triggerModel.create("tm")
triggerModel.addblock.source.output("tm", "output-on", 1, 1)
-- Step to the next source voltage in the sweep.
triggerModel.addblock.source.action.step("tm", "step-voltage", 1)
-- Delay for 200 ms.
triggerModel.addblock.delay.constant("tm", "delay", 200e-3)
-- Make a current measurement.
triggerModel.addblock.measure("tm", "measure", 1, 1)
-- Loop to the voltage step.
triggerModel.addblock.branch.counter("tm", "branch", "step-voltage", 11)
triggerModel.addblock.source.output("tm", "output-off", 1, 0)
-- Initiate the trigger model.
triggerModel.initiate("tm")
```

SMU pulse sweep example

Parameters

- SMU slot 1, channel 1
- Linear pulse sweep 0 V to 10 V
- 0 V bias
- 100 ms pulses

Code

```
-- Alias the SMU.
local smu = slot[1].smu[1]
-- Configure linear sweep
smu.trigger.source.linearv(0,10,11)
-- Set bias level to 0 V.
smu.source.levelv = 0
-- Alias trigger model
local triggerModel = slot[1].trigger.model
triggerModel.create("tm")
```

```
triggerModel.addblock.source.output("tm", "output-on", 1, 1)
-- Step to next source voltage in sweep
triggerModel.addblock.source.action.step("tm", "step-voltage", 1)
triggerModel.addblock.delay.constant("tm", "delay1", 100e-3)
-- Set source voltage to 0V
triggerModel.addblock.source.action.bias("tm", "bias-zero", 1)
triggerModel.addblock.delay.constant("tm", "delay2", 100e-3)
-- Loop back to voltage step
triggerModel.addblock.branch.counter("tm", "branch", "step-voltage", 11)
triggerModel.addblock.source.output("tm", "output-off", 1, 0)
-- Initiate trigger model
triggerModel.initiate("tm")
```

SMU pulse sweep (EOA) example

Parameters

- MSMU200-2 slot 1, channel 1
- Linear pulse sweep 1 A to 5 A (pulse-only region)
- Measure current and voltage once during each step
- 0 A bias
- 250 µs pulse width, 5% duty cycle
- 4-wire connections to DUT

Code

```
local smu = slot[1].smu[1]
local pulsewidth = 250e-6
local measure_aperture = 50e-6
local duty_cycle = 0.05  -- 5%
local tm_delay_time = pulsewidth - measure_aperture
local tm_toff_time = (pulsewidth / duty_cycle) - pulsewidth
local current_range = 7
local voltage_range = 20
local start_i = 1
local stop_i = 5
local numPulses = 5
-- Set bias level
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.rangei = current_range
smu.source.leveli = 0
smu.source.limitv = voltage_range
smu.source.pulse.limitv = voltage_range
-- Use remote sense for high current measurements
smu.sense = smu.SENSE_4WIRE
-- Set fixed ranges and measure aperture
smu.measure.rangei = current_range
smu.measure.rangev = voltage_range
smu.measure.autorangei = smu.OFF
smu.measure.aperture = measure_aperture
-- Configure linear sweep and what to measure
smu.trigger.source.lineari(start_i, stop_i, numPulses)
smu.trigger.measure.iv(smu.defbuffer1, smu.defbuffer2)
-- Clear and prepare buffers for data collection
smu.defbuffer1.clear()
smu.defbuffer1.appendmode = 1
smu.defbuffer2.clear()
smu.defbuffer2.appendmode = 1
-- Alias trigger model
local triggerModel = slot[1].trigger.model
-- If run more than once, need to delete first
triggerModel.delete("tm")
triggerModel.create("tm")
-- Output on at beginning of trigger model
triggerModel.addblock.source.output("tm", "output-on", 1, 1)
-- Step to next source current in sweep
triggerModel.addblock.source.action.pulsestep("tm", "step-current", 1)
-- Delay relative to the step-current block
```

```
triggerModel.addblock.delay.constant("tm", "delay1", tm_delay_time, "step-current")
-- Measure one time from channel 1
triggerModel.addblock.measure("tm", "measure", 1, 1)
-- Set bias condition
triggerModel.addblock.source.action.bias("tm", "bias-zero", 1)
-- Off time
triggerModel.addblock.delay.constant("tm", "delay2", tm_toff_time)
-- Loop back to current step for next pulse
triggerModel.addblock.branch.counter("tm", "branch", "step-current", numPulses)
-- Output off at end of trigger model
triggerModel.addblock.source.output("tm", "output-off", 1, 0)
-- Initiate trigger model
triggerModel.initiate("tm")
waitcomplete()
        print("Time\t\t IF\t\t VF")
        for j = 1, smu.defbuffer1.n , 1 do
                print(smu.defbuffer1.timestamps[j], smu.defbuffer1[j],smu.defbuffer2[j])
        end
        print("limitV ".. smu.source.limitv)
        print("meas_range_V ".. smu.measure.rangev)
        print("pulse_limit_V ".. smu.source.pulse.limitv)
```

Trigger models

Trigger models

TriggerFlow trigger models control the sequence in which source and measure actions occur. Trigger models can also be used to synchronize actions with other events in the test system, both internal and external. They are optimized for timing control and execution speed, and should be used for any test flow that requires precise timing or many actions that must occur in a particular order. You can define any number of trigger models for a module slot.

Trigger models control the flow of a test routine using a combination of Trigger model blocks (on page 69) and Events (on page 76).

TriggerFlow trigger models allow you to fully customize a trigger model for your test sequence. The following are examples of actions you can create within your trigger model:

- Wait for an event to occur before the instrument makes another measurement
- Notify other equipment and timers that an event has occurred
- Wait for another piece of equipment to signal completion
- Use configuration lists to sequence different source and measure settings dynamically during trigger model operation
- Specify delays between events and measurements
- Create pulses
- Store measurements into a given buffer until an event occurs
- Use branching to create loops and conditions
- Step through pre-configured linear, log, and list sweeps

When running repeated tests, see Load or unload a trigger model (on page 88) for additional timing optimization.

Trigger model blocks

Each trigger model consists of blocks that can be combined to create the trigger model. Blocks are added to a trigger model using remote commands. Each block includes a name parameter that can be used to reference that block from other blocks. If a block is unnamed, it cannot be referenced. Depending on the block, additional parameters are required to configure the block. A trigger model must include at least one block that performs an operation on a specific channel before it can be run.

A module installed in the MP5103 can support multiple trigger models. Each trigger model may contain blocks that operate on one or more channels, provided that all channels used in the block are on the module where the trigger model has been defined.

Channels can only be controlled by one running trigger model at a time. Multiple trigger models may run simultaneously as long as they do not use the same channels. For example, if "TriggerModelA" is running on a module installed in slot 1 and includes a block that operates on channel 1, no other trigger model using channel 1 can be started on that module until "TriggerModelA" completes or is aborted.

You can combine trigger model blocks as you would construct a flowchart diagram. Trigger models are created using these fundamental block categories:

- **Timing:** Waits for an event or a defined delay to occur before the flow continues
- **Action:** Starts an action in the instrument, such as making a measurement
- **Notify:** Notifies other equipment, timers, or the user that an event has occurred
- **Branch:** Branches to another block when a condition has been satisfied

Each type of block is described in the following topics.

Timing blocks

You can use the timing blocks to control the timing of actions in the trigger model. Timing blocks include the wait and delay blocks.

Constant delay block

When trigger model execution reaches a delay block, it stops the trigger model operation for the time set by the delay. Background measurements continue to be made.

This delay waits for the delay time to elapse before proceeding to the next block in the trigger model.

The delay block has an optional reference parameter that you can use to mark the start of the delay. When using a reference, you must identify the trigger block at which the delay should start. The reference parameter creates very precise delays, such as when configuring the source to output a pulse waveform. Using the reference parameter allows for the most consistent pulse timing because the start of the delay does not depend upon the completion of any block between the reference and the delay blocks. If the use of a reference parameter causes the delay time to elapse before the delay block executes, the delay block effectively executes a zero-second delay.

The delay list is also optional and consists of a table of delay times, in seconds, to iterate through each time the delay block is encountered. You can use this to sequence through different delay times in a sweep.

Use the following TSP command to implement a constant delay block. Select to view command details.

slot[Z].trigger.model.addblock.delay.constant() (on page 587)

Wait block

The wait block causes the trigger model to stop and wait for an event or set of events to occur before continuing. You can specify up to four events for each wait block, and the logic (AND or OR) to combine them. See Events (on page 76) for more details.

The event can occur before the trigger model reaches the wait block. If the event occurs after the trigger model starts but before the trigger model reaches the wait block, the trigger model records the event. By default, when the trigger model reaches the wait block, it executes the wait block without waiting for the event to happen again (the clear parameter is set to never).

If the clear parameter is set to enter, previously-recorded events will be cleared and the block will wait for a new event. The instrument clears the memory of the recorded event when the trigger model exits the wait block.

All events in the list are subject to the same action. You cannot combine `AND` with `OR` logic in a single wait block.

A wait block can be used at the beginning of several trigger models to coordinate all of them to start simultaneously after a single event. This can be combined with interactive triggering. See Interactive triggering (on page 85) for more information.

Use the following TSP command to implement a wait block. Select to view command details.

slot[Z].trigger.model.addblock.wait() (on page 614)

Action blocks

Action blocks start an action on a channel, such as making a measurement or changing instrument source and measure attributes. Action blocks include the following trigger model blocks:

- Configuration list
- Measure
- Measure overlapped
- No operation
- Source action
- Source output

Configuration list blocks

You can use configuration list blocks to recall settings that are stored in a configuration list. When the trigger model reaches a configuration list block, the settings in the list are applied.

The trigger model blocks that recall configuration lists are:

- Configuration next
- Configuration previous
- Configuration recall

Configuration list next block

When trigger model execution reaches a configuration recall next block, the settings at the next index in the specified configuration list are restored.

The first time the trigger model encounters this block for a specific configuration list, the first index is recalled. Each subsequent time this block is encountered, the settings at the next index in the configuration list are recalled and applied before the next block executes. When the last index in the list is reached, it returns to the first index.

Use the following TSP command to implement a configuration list next block. Select to view command details.

slot[Z].trigger.model.addblock.configlist.next() (on page 580)

Configuration list recall previous block

When trigger model execution reaches a configuration recall previous block, the settings at the previous index in the specified configuration list are restored.

Each subsequent time trigger model execution reaches this block, the configuration list goes backward one index. If the configuration list has not been accessed previously in the model, the first execution of a configuration list previous block loads the last index in the list.

Use the following TSP command to implement a configuration list previous block. Select to view command details.

slot[Z].trigger.model.addblock.configlist.prev() (on page 582)

Configuration list recall block

When the trigger model reaches a configuration recall block, the settings in the specified configuration list are recalled.

You can restore a specific set of configuration settings in the configuration list by defining the index.

Use the following TSP command to implement a configuration list recall block. Select to view command details.

slot[Z].trigger.model.addblock.configlist.recall() (on page 585)

Measure block

This block triggers measurements based on the measurements configured by slot[Z].psu[X].trigger.measure.Y() (on page 403) or slot[Z].smu[X].trigger.measure.Y() (on page 530). Create and configure reading buffers prior to initiating the trigger model. Measurement settings can be configured before initiation or by recalling a configuration list during trigger model execution. When trigger model execution reaches this block:

- The channel begins making measurements.
- The trigger model execution waits for the measurement to be made.
- The instrument processes the reading and places it into the reading buffer specified by slot[Z].psu[X].trigger.measure.Y() (on page 403) or slot[Z].smu[X].trigger.measure.Y() (on page 530). The reading buffer must be created or assigned before executing the trigger model.

Set the measure count in the trigger block. If no measure count is specified, the instrument will use slot[Z].psu[X].measure.count (on page 382) or slot[Z].smu[X].measure.count (on page 496). Trigger model execution does not proceed until all measurements are complete.

Use the following TSP command to implement a measure block. Select to view command details.

slot[Z].trigger.model.addblock.measure() (on page 590)

Measure overlapped block

This block is only for SMU instruments. It triggers measurements based on the measurements configured by slot[Z].smu[X].trigger.measure.Y() (on page 530). Create and configure reading buffers prior to initiating the trigger model. Measurement settings can be configured before initiation or by recalling a configuration list during trigger model execution. When trigger model execution reaches this block:

- The instrument begins triggering measurements.
- The instrument processes the reading and places it into the reading buffer specified by slot[Z].smu[X].trigger.measure.Y() (on page 530). The reading buffer must be created or assigned before executing the trigger model.
- The trigger model proceeds with the remaining trigger model blocks while measurements are being made.

If there is one measure overlapped block for a channel, there may be no other measure or measure overlapped blocks for that channel.

Use the following TSP command to implement a measure overlapped block. Select to view command details.

slot[Z].trigger.model.addblock.measureoverlapped() (on page 592) (SMUs only)

No operation block

This block acts as a placeholder in a trigger model. No action is taken when this block is executed. A named, no operation block can be used with a branch block as a reference place to which the trigger model can branch.

Use the following TSP command to implement a no operation block. Select to view command details.

slot[Z].trigger.model.addblock.nop() (on page 594)

Source action blocks

Source action blocks change the source level according to the sweep configuration. The sweep configuration must be defined prior to trigger model initiation using the following functions:

- slot[Z].psu[X].trigger.source.linearY() (on page 405) | slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].psu[X].trigger.source.listY() (on page 406) | slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].psu[X].trigger.source.logY() (on page 408) | slot[Z].smu[X].trigger.source.logY() (on page 535)

Where:

`Y` = Source function; for SMUs, `v` = voltage, `i` = current; for PSUs, only voltage can be used.

Source functions cannot be changed within a sweep.

Source action bias block

The source action bias block sets the source output level to a level set before the start of the trigger model using the `level` command. For PSU modules, see slot[Z].psu[X].source.levelv (on page 392). For SMU modules, see slot[Z].smu[X].source.levelY (on page 514).

This block can be used with source action step (on page 73) blocks to create a pulse sweep that returns to the bias level after each sweep step.

Use the following TSP command to implement a source action bias block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)

Source action set block

The source action set block sets the output level to the value of the present index of the sweep table without advancing the index. This can be used to reapply a source value or hold a source value for a certain time before proceeding with the sweep.

The sweep table is generated using the linear, list, or log sweep configuration commands before the trigger model is created and run.

- slot[Z].psu[X].trigger.source.linearY() (on page 405) | slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].psu[X].trigger.source.listY() (on page 406) | slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].psu[X].trigger.source.logY() (on page 408) | slot[Z].smu[X].trigger.source.logY() (on page 535)

For more information on configuring sweeps, see Configuration lists and sweeps (on page 50).

Use the following TSP command to implement a source action set block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.set() (on page 606)

Source action skip block

The skip source action block is used to advance the index of the sweep table without changing the source level.

The sweep table is generated using the linear, list, or log sweep configuration commands before the trigger model is created and run.

- slot[Z].psu[X].trigger.source.linearY() (on page 405) | slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].psu[X].trigger.source.listY() (on page 406) | slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].psu[X].trigger.source.logY() (on page 408) | slot[Z].smu[X].trigger.source.logY() (on page 535)

For more information on configuring sweeps, see Configuration lists and sweeps (on page 50).

Use the following TSP command to implement a source action skip block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.skip() (on page 608)

Source action step block

This block advances a sweep to the next value in the sweep table and changes the source output to the level specified in the configured sweep.

The sweep table is generated using the sweep configuration commands before the trigger model is created and run.

- slot[Z].psu[X].trigger.source.linearY() (on page 405) | slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].psu[X].trigger.source.listY() (on page 406) | slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].psu[X].trigger.source.logY() (on page 408) | slot[Z].smu[X].trigger.source.logY() (on page 535)

For more information on configuring sweeps, see Configuration lists and sweeps (on page 50).

The source action step block can be used with the Source action bias (on page 72) block to create a pulse sweep that returns to a bias voltage after each sweep point.

Use the following TSP command to implement a source action step block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.step() (on page 610)

Source action pulse set block

The source action pulse set block sets the pulse output level to the value of the present index of the sweep table without advancing the index. This can be used to reapply a source value or hold a source value for a certain time before proceeding with the sweep. This command must be used to access extended operating area pulse levels.

The sweep table is generated using the linear, list, or log sweep configuration commands before the trigger model is created and run.

- slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].smu[X].trigger.source.logY() (on page 535)

For more information on configuring sweeps, see Configuration lists and sweeps (on page 50).

Use the following TSP command to implement a source action pulse set block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.pulseset() (on page 602)

Source action pulse step block

This block advances a sweep to the next value in the sweep table and changes the source output pulse to the level specified in the configured sweep. This command must be used to access extended operating area pulse levels.

The sweep table is generated using the sweep configuration commands before the trigger model is created and run.

- slot[Z].smu[X].trigger.source.linearY() (on page 532)
- slot[Z].smu[X].trigger.source.listY() (on page 533)
- slot[Z].smu[X].trigger.source.logY() (on page 535)

For more information on configuring sweeps, see Configuration lists and sweeps (on page 50).

The source action step pulse block can be used with the Source action bias (on page 72) block to create a pulse sweep that returns to a bias voltage after each sweep point.

Use the following TSP command to implement a source action pulse step block. Select to view command details.

slot[Z].trigger.model.addblock.source.action.pulsestep() (on page 604)

Source output block

The source output block turns the output of the source on or off.

This block does not determine the settings of the output source (such as the output-off mode and output-off current limit). The source settings are determined by either the present settings of the instrument or by a configuration list. This block should not be used when a measure overlapped block is running.

Use the following TSP command to implement a source output block. Select to view command details.

slot[Z].trigger.model.addblock.source.output() (on page 612)

Notify blocks

Notify blocks generate events using the trigger model. Events generated are unique to the module running the trigger model. This is helpful for controlling external equipment or coordinating actions between trigger models on different slots. These blocks complete the action of sending a notification and then moving to the next trigger model block.

Notify blocks include the following trigger model blocks:

- Log event
- Notify

Log event block

This block allows you to log an event in the event log when trigger model execution reaches this block. This may be useful for debugging a trigger model, monitoring progress, or posting results.

You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

You can define the type of event (information, warning, abort model, or error). All events generated by this block are logged in the event log.

Note that using this block too often in a trigger model could overflow the event log. The log event block takes additional time to record the event and may impact timing in critical paths. Avoid using this block in time-sensitive sections of the trigger model, such as when there must be a precise delay between two trigger model blocks or actions.

Use the following TSP command to implement a log event block. Select to view command details.

slot[Z].trigger.model.addblock.logevent() (on page 589)

Notify block

This block generates a trigger event in the system. You can use this event to trigger actions elsewhere in the system. For example, you can configure the notify event as the stimulus to generate an output trigger on a digital I/O line or to start an internal timer.

Notify blocks require a notify event number to be set.

See Configure trigger events (on page 79) for more information.

Use the following TSP command to implement a notify block. Select to view command details.

slot[Z].trigger.model.addblock.notify() (on page 596)

Branch blocks

A branch block conditionally jumps to a trigger block other than the next block in the trigger model. For example, if you need to set up the trigger model to wait for an event under some conditions but not others, you can use a branch block to define when the wait block is enabled. You can use a once block (on page 76) to create a bypass and skip the wait block the first time the trigger model runs. This makes it possible to avoid deadlock when multiple instruments are being synchronized and each one is waiting for notification from the other one to start the trigger model.

Branch blocks include the following trigger model blocks:

- Always
- Loop or branch counter
- On event
- Once
- Once excluded
- Reset branch counter

Always block

When the trigger model reaches a branch-always block, it always proceeds to the block that you specified.

Use the following TSP command to implement a branch-always block. Select to view command details.

slot[Z].trigger.model.addblock.branch.always() (on page 571)

Loop or branch counter block

When trigger model execution reaches a loop counter block, it goes to a specified block until the count value is reached. When the counter exceeds the count value, trigger model execution ignores the branch, continues to the next block in the sequence, and resets the counter.

The counter is reset to 0 when the trigger model starts and is incremented immediately before the branch compares the actual counter value to the set counter value. Therefore, the counter is at 0 until the first branch block is reached. The counter then increments to 1 and is compared to the count value. This causes the branch counter to behave similar to a do-while loop; the number of actual branches is count-1. The count parameter should be set to the number of times you want a section of the trigger model to execute.

When the trigger model reaches the set counter value, branching stops and the counter value is one greater than the setting.

Use the following TSP command to implement a loop counter block. Select to view command details.

slot[Z].trigger.model.addblock.branch.counter() (on page 572)

On event block

The branch-on-event block branches to a specified block when a specified trigger event occurs. If the trigger event has not yet occurred when trigger model execution reaches the branch-on-event block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the next time trigger model execution reaches the branch-on-event block, it goes to the branching block.

If you set the branch event to none, an error is generated when you run the trigger model.

Trigger events are reset when the trigger model is at the start block, so only events that occur after you start trigger model execution are detected by the branch-on-event block. The event is also reset after trigger model execution completes the branching block. See Events (on page 76) for additional details.

Use the following TSP command to implement a branch-on-event block. Select to view command details.

slot[Z].trigger.model.addblock.branch.event() (on page 574)

Once block

When the trigger model reaches a branch-once block, it goes to a specified block the first time it is encountered in the trigger model. If it is encountered again, the trigger model ignores the block and continues in the normal sequence.

You can use this block to create a bypass. For example, you might place a branch-once block before a wait block to skip the wait block on the first pass of the trigger model.

The once block is reset when the trigger model reaches the idle state. Therefore, the branch-once block will always execute the first time the trigger model encounters this block.

Use the following TSP command to implement a branch once block. Select to view command details.

slot[Z].trigger.model.addblock.branch.once() (on page 576)

Once excluded block

The branch-once-excluded block is ignored the first time the trigger model encounters it. After the first encounter, the trigger model goes to the specified branching block.

The branch-once-excluded block is reset when the trigger model starts or is placed in idle.

Use the following TSP command to implement a branch-once-excluded block. Select to view command details.

slot[Z].trigger.model.addblock.reset.branch.onceexcluded() (on page 577)

Reset branch count block

When this block is encountered in the trigger model, the specified branch counter resets to 0.

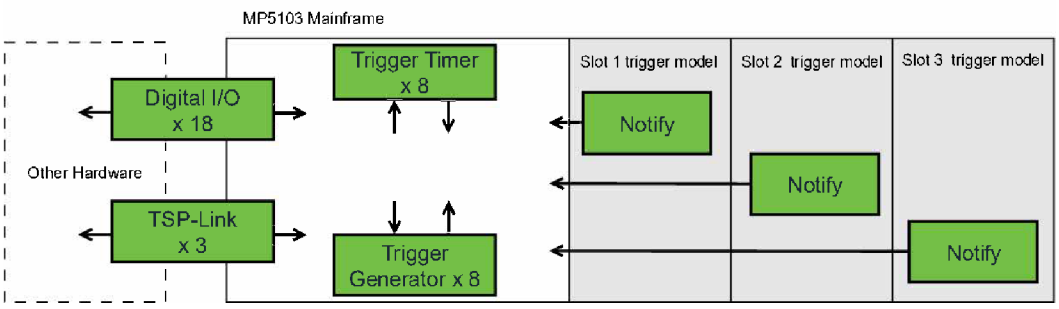
Use the following TSP command to implement a reset branch count block. Select to view command details.

slot[Z].trigger.model.addblock.reset.branch.counter() (on page 597)

Events

Trigger model events are responses to actions within a test system. Events can be external or internal. They can also be directional. For example, an event ID can be used to indicate an incoming event to detect or an outgoing event to generate.

Events are referred to by ID constants. Some event types may have multiple constants depending on the line used. For example, pin or line 1 for digital I/O has a different ID from pin or line 2.



Events are most commonly used within a trigger model. Trigger model blocks such as branch-on-event and wait blocks use events to control the flow of the test program. The events may be generated from actions taken by the system, such as executing a notify block, or they may be the result of trigger objects that provide additional functionality. Trigger objects include:

- Digital I/O pins
- TSP-Link trigger lines
- Internal events (including timers and trigger generators and detectors)

You must configure an event before it can be acted upon in the trigger model; see Configure trigger events (on page 79) for more information. Events can also be used outside of the trigger model. For more information, see Interactive triggering (on page 85).

External events

External trigger model events are responses to actions external to a test system.

You can use the digital I/O lines and TSP-Link synchronization lines to synchronize the operations of the MP5103 with those of external instruments. You can use TSP-Link lines to synchronize the MP5103 with other TSP-enabled instruments, including other MP5103 instruments. You must use the digital I/O lines to synchronize the MP5103 with products that do not support TSP-Link.

TSP-Link and Digital I/O lines are configured and controlled similarly. See Digital I/O (on page 77) and TSP-Link (on page 106) for more information about connections, configuration, and control of the lines.

Digital I/O

The MP5103 has a digital input/output port with 18 lines that can control external digital circuitry. When digital I/O lines are configured for triggering, events can be generated or detected.

The event ID for digital I/O is digio.trigger[N].EVENT_ID (on page 146). See MP5000 Series commands (on page 129) for more information.

This ID can be used to refer to an event generated on the trigger line `N`. Event IDs are generated when digital I/O lines change state according to the configured trigger mode. Trigger model blocks that use events can use this ID to make the block dependent on a digital I/O event. Another object can respond to this event outside of the trigger model by setting the stimulus attribute to this event ID.

Digital I/O lines must be configured for triggering before creating the trigger model. For more information, see Digital I/O trigger control modes (on page 79).

TSP-Link

TSP-Link is a high-speed trigger synchronization and communication bus that test system builders can use to connect multiple compatible instruments in a master and subordinate configuration.

The tsplink.trigger[N].EVENT_ID (on page 322) constant can be used to refer to a trigger detected on the TSP-Link trigger line *N*. Trigger model blocks that use events can use this ID to make the block dependent on a TSP-Link event. Another object can respond to this event outside of the trigger model by setting the stimulus attribute to this event ID.

Three TSP-Link trigger lines can be used for triggering, but they must be configured before being used in a trigger model. See TSP-Link and TSP-Net (on page 106) for more information on controlling trigger lines.

Internal events

Internal events originate from the MP5103 itself. These events are useful for monitoring time outside of the trigger model or using an internal trigger generator.

Limit events

A limit event is generated when a SMU module reaches the current or voltage limit. This limit is set before trigger model initiation or by recalling a configuration list. Limit events are useful for detecting changes in device behavior, such as stressing a device until breakdown and then taking action in the trigger model.

The event ID for a limit event is slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529). This event is only applicable to SMU modules. Each channel can generate an event independently.

Notify events

A notify event is generated when a trigger model executes a notify block. Notify events are useful for synchronization and generation of other events from the trigger model. For example:

- A notify block in a trigger model can be used as the stimulus for a digital I/O line
- A wait block in one trigger model can wait for a notify event from a separate trigger model
- A notify block can be used to start a trigger timer to run during execution

Notify events can be used to indicate events unique to a single module in the test system. The MP5103 PSU modules have 16 unique notify event IDs and the SMU modules have 8 unique notify event IDs.

The event ID for a notify event is slot[Z].trigger.model.EVENT_NOTIFYN (on page 618), where `N` is the number of the notify event. Specify the event ID in the notify block to indicate which event is generated when the block executes. The same ID can also be set as the stimulus for other trigger model blocks or trigger objects.

Timer events

The MP5103 contains 8 individual timers for monitoring processes. An event is generated when the timer's delay expires. Additionally, the timer can be configured to generate an event when it is triggered to begin and to restart the timer after expiration to generate repeating, regular events.

The event ID for a timer event is trigger.timer[N].EVENT_ID (on page 310), where *N* is the number of the timer, 1 to 8. This event ID can be set as the stimulus for other trigger objects and in the trigger model. The timer must be configured before being used to generate events. For more information see Trigger timers (on page 83).

Configuring internal trigger events

The MP5103 has eight trigger generators available from each module that you can use to generate trigger events.

The trigger.generator[N].EVENT_ID (on page 308) constant is an identification number that identifies events generated by this generator. To have another trigger object respond to trigger events generated by this generator, set the stimulus attribute of the other object to the value of this constant.

Event blenders

The ability to combine trigger events into a single trigger event output is called event blending. You can use an event blender to wait for up to four input trigger events to occur before responding with an output event.

Event blenders are useful for applications such as:

- Waiting for multiple external instruments to trigger digital I/O lines before moving to another step
- Synchronizing multiple notify events from multiple trigger models in the MP5103
- Sending an output trigger when one or more trigger models branches to a notify that a condition has occurred

You can program up to four event blenders for the MP5103.

Trigger event IDs

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Configure trigger events

Some events require additional configuration. These events include:

- Digital I/O
- TSP-Link
- Trigger timers
- Trigger blenders

You must configure trigger events before the trigger model is initiated. For example, if you want the trigger model to wait for a trigger on digital I/O line 1, then line 1 must be configured as an input line. The following sections will cover how to configure each event source.

Digital I/O and TSP-Link trigger control modes

Digital I/O and TSP-Link lines configured with a trigger control mode generate events that can be used in the trigger model or for interactive triggering. These modes either detect or generate transitions in the state of the line, from high to low (falling edge) or from low to high (rising edge). The input edge detection setting of the line determines which type of transition is detected as an input trigger. Output triggers are typically generated in the form of a pulse. The type of transition that occurs on the leading edge of the pulse is determined by an output logic setting for each line. The duration of the output pulse is determined by a pulse width.

There are three options to configure a digital I/O line or TSP-Link line for triggering:

- Open-drain
- Trigger input or output only
- Synchronous triggering

Trigger output or input only

These modes automatically set up a line for a single operating mode. Lines can either generate triggers or receive triggers.

Output mode

When you place a line in output mode, it is automatically set high or low depending on the output logic setting. Use the negative logic setting when you want to generate a falling edge trigger. Use the positive logic setting when you want to generate a rising edge trigger. You cannot detect incoming triggers on a line configured as a trigger output.

To set a line to be an output-only line:

1. Send `digio.trigger[N].reset()` or `tsplink.trigger[N].reset()` to reset the line.
2. Send `digio.trigger[N].mode = digio.MODE_TRIGGER_OUT` or `tsplink.trigger[N].mode = tsplink.MODE_TRIGGER_OUT`.
3. Set the output trigger logic to positive or negative using the `digio.trigger[N].logic` or `tsplink.trigger[N].logic` command.
4. Configure trigger pulse width with the `digio.trigger[N].pulsewidth` or `tsplink.trigger[N].pulsewidth` command.
5. Identify the event that triggers the digital output trigger by configuring the `digio.trigger[N].stimulus` or `tsplink.trigger[N].stimulus` commands.

Input mode

When you place a line in input mode, it is automatically set high to allow it to detect and respond to externally generated triggers. Depending on the input edge detection setting, the line can detect falling-edge triggers, rising-edge triggers, or both.

The line cannot generate an output trigger if it is set to the trigger input mode.

To set a line to be an input-only line:

1. Send `digio.trigger[N].reset()` or `tsplink.trigger[N].reset()` to reset the line.
2. Send `digio.trigger[N].mode = digio.MODE_TRIGGER_IN` or `tsplink.trigger[N].mode = tsplink.MODE_TRIGGER_IN`.
3. Set the input trigger edge detection type to falling, rising, or other by using the `digio.trigger[N].edge` or `tsplink.trigger[N].edge` command. You can also generate a trigger output immediately at the digital I/O line using `digio.trigger[N].assert()`.

Open drain

When you set the digital I/O line to an open-drain trigger mode, the line is configured to be an open-drain signal with a 100 kΩ pull-up resistor. This makes the line compatible with other instruments that use open-drain trigger signals. In this mode, you can use the line to detect input triggers or generate output triggers, or both. To use this mode successfully, you must carefully configure the input edge and output logic settings because both of these affect the initial state of the trigger line. It is recommended that you reset the line before implementing and configuring this mode.

To use the open-drain trigger line only as a trigger input:

1. Send `digio.trigger[N].reset()` or `tsplink.trigger[N].reset()` to reset the line.
2. Send `digio.trigger[N].mode = digio.MODE_TRIGGER_OPEN_DRAIN` or `tsplink.trigger[N].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN`.
3. Set the input trigger edge detection type to negative (falling edge) or positive (rising edge) with the `digio.trigger[N].edge` or `tsplink.trigger[N].edge` command.

The command that sets the edge type low automatically sets the line high. This enables the line to respond to and detect externally generated triggers.

Do not set the output trigger logic type to positive after setting the edge detection type. This sets the line low, which will prevent the line from operating correctly as a trigger input.

To use the open-drain trigger line only as a trigger output:

1. Send `digio.trigger[N].reset` or `tsplink.trigger[N].reset()` to reset the line.
2. Send `digio.trigger[N].mode = digio.MODE_TRIGGER_OPEN_DRAIN` or `tsplink.trigger[N].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN`.
3. Set the output trigger logic type to negative (falling edge) or positive (rising edge) with the `digio.trigger[N].logic` or `tsplink.trigger[N].logic` command.
4. Configure trigger pulse width with the `digio.trigger[N].pulsewidth` or `tsplink.trigger[N].pulsewidth` command.
5. Set the stimulus attribute to configure the event that will generate an output trigger.

When you set the logic type to negative, the instrument automatically sets the line high. Setting the logic type to positive automatically sets the line low.

Do not set the input trigger edge detection type after setting the positive logic type. This will set the line high, which will prevent the line from operating correctly as a trigger output.

To use the line as both a trigger input and a trigger output (falling edge triggers only):

1. Send `digio.trigger[N].reset()` or `tsplink.trigger[N].reset()` to reset the line.
2. Send `digio.trigger[N].mode = digio.MODE_TRIGGER_OPEN_DRAIN` or `tsplink.trigger[N].mode = tsplink.MODE_TRIGGER_OPEN_DRAIN`.
3. Set the output trigger logic type to negative (falling edge) with the `digio.trigger[N].logic` or `tsplink.trigger[N].logic` command.
4. Set the input trigger edge detection type to falling, rising, or either using the `digio.trigger[N].edge` or `tsplink.trigger[N].edge` command.
5. Configure trigger pulse width with the `digio.trigger[N].pulsewidth` or `tsplink.trigger[N].pulsewidth` command.
6. Configure the stimulus to generate a trigger from another event with the `digio.trigger[N].stimulus` or `tsplink.trigger[N].stimulus` command.

You can use these settings for digital I/O line triggering applications that use Tektronix products offering Trigger Link.

Synchronous triggering

The synchronous triggering modes allow you to:

- Implement bidirectional triggering on a single trigger line
- Start operations on one or more external instruments using a single trigger line
- Wait for all instruments to complete all triggered actions

This mode is best used when actions must be synchronized to start and all actions must be completed. The master is responsible for starting the operations and detecting when all other instruments have completed. There should only be one synchronous master in a triggering system. The acceptors are responsible for holding their trigger line low until they are ready for the next task.

Instruments that are configured as synchronous acceptors detect the falling-edge trigger and begin their triggered actions. At the same time, they latch the line low and hold it in that state until their triggered actions complete. Each instrument configured as an acceptor releases the line upon completion of its triggered actions.

When all instruments have released the line, the line changes state and generates a rising edge trigger. This trigger is detected by the synchronous master, which then performs its next triggered action.

Synchronous master

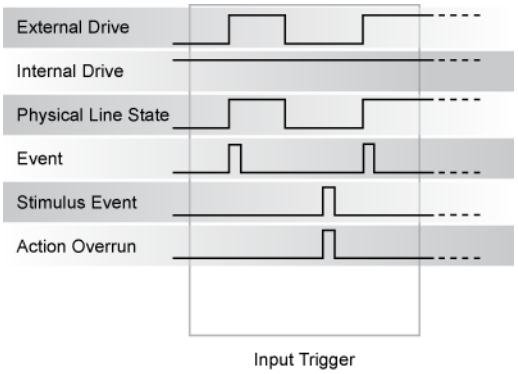
When a line is set to the synchronous master mode, it generates falling edge output triggers and detects rising edge input triggers on the same trigger line.

Input characteristics

All rising edges are input triggers.

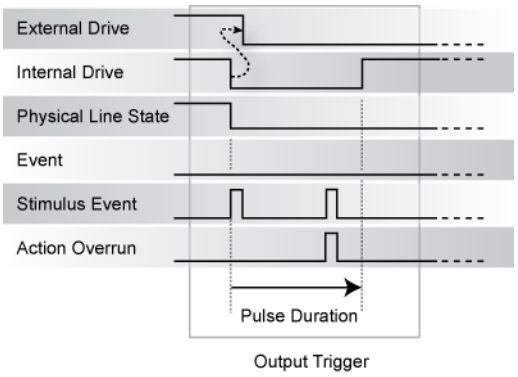
When all external drives or synchronous acceptors release the physical line, the rising edge is detected as an input trigger.

A rising edge is not detected until all external drives release the line and the line floats high.



Output characteristics

- In addition to trigger events from other trigger objects, the TSP command `digio.trigger[N].assert()` or `tsplink.trigger[N].assert()` generates a low pulse that is similar to the falling-edge trigger mode.
- An action overrun occurs if the physical line state is low when a stimulus event occurs.

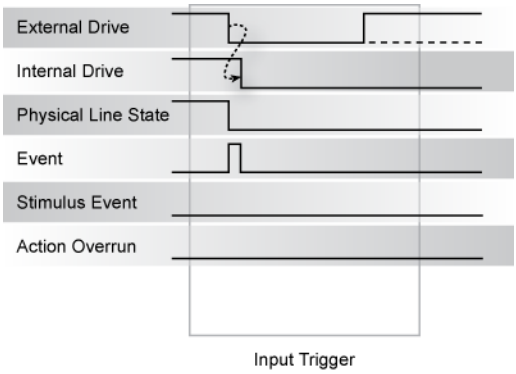


Synchronous acceptor

A line that is set to the synchronous acceptor mode detects falling edge input triggers and generates rising edge output triggers on the same trigger line. When a line that is configured as synchronous acceptor detects the falling edge trigger, it latches the line low and holds it in that state until all triggered actions for that instrument are complete. When the triggered actions are complete, the synchronous acceptor line releases the line. When all connected instruments have released the line, the line changes state and generates a rising edge trigger.

Input characteristics

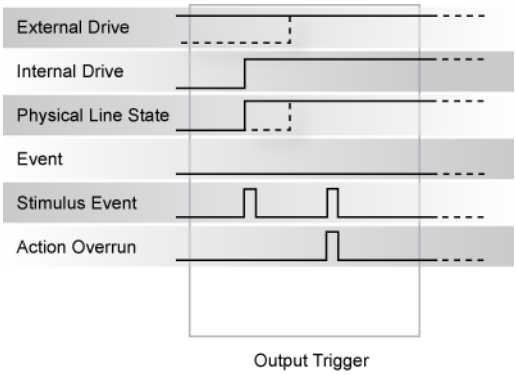
The falling edge is detected as the external drive pulses the line low, and the internal drive latches the line low.



Output characteristics

In addition to trigger events from other trigger objects, the TSP command `digio.trigger[N].assert()` or `tsplink.trigger[N].assert()` also triggers events.

- The physical line state does not change until all drives (internal and external) release the line.
- Action overruns occur if the internal drive is not latched low and a source event is received.



Trigger timers

Trigger timers are used to track time independently from the trigger model, unlike delay blocks, which reside inside of the trigger model. Timers can be started by other events using the stimulus attribute. Timer events can be generated at different times depending on configuration.

The MP5103 has eight independent timers. Configure timers before using their events to start other actions.

Timer delay time

The delay value of the timer is the period after the timer is triggered and before the timer generates a trigger event. All delay values are specified in seconds.

The delay must be configured prior to the trigger model using the trigger.timer[N].delay (on page 310) command.

Timer event count

The count sets the number of times the timer delays. Each time the timer delay elapses, it generates a trigger event (trigger.timer[N].event_ID (on page 310)).

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

The count is set using trigger.timer[N].count (on page 309).

Timer stimulus

Configure which event triggers the start of the timer using the trigger.timer[N].stimulus (on page 313) attribute. You cannot start a trigger timer using a command; timers can only be started using other trigger events.

Pass-through mode

When enabled, the timer generates a trigger event immediately when it is started. The timer generates additional trigger events each time a delay expires. If the pass-through mode is disabled, the timer does not generate a trigger event until after the first delay elapses. Use trigger.timer[N].passthrough (on page 311) to configure the pass-through mode.

Trigger generators and detectors

The MP5103 has eight trigger generators available that you can use to generate trigger events. No additional configuration is needed to use the trigger generators. The trigger.generator[N].EVENT_ID (on page 308) constant is an identification number that identified events generator by generator N. The most common use for trigger generators is manually triggering from a command interface or a script with the trigger.generator[N].assert() (on page 307) function. For more information, see Interactive triggering (on page 85).

The MP5103 also has eight trigger detectors. The detectors can be configured to detect other events using the trigger.detector[N].stimulus (on page 305) attribute.

Event blenders

Event blenders allow you to combine multiple different event inputs into a single output event. Use blenders when you need to wait for multiple triggers to proceed or when you need to detect one of many triggers.

The MP5103 has four trigger blender events. Each event can monitor up to four trigger stimuli.

Event blender modes

Event blenders perform logical AND and logical OR functions on trigger events. For example, trigger events can be triggered when either a manual trigger or external input trigger is detected.

- **Or:** Generates an event when an event is detected on any one of the four stimulus inputs
- **And:** Generates an event when an event is detected on all of the assigned stimulus inputs

Set the `trigger.blender[N].orenable` attribute to configure the event blender mode. Setting the attribute to `true` enables OR mode; setting the attribute to `false` enables AND mode.

Assigning input trigger events

Each event blender has up to four stimulus inputs. You can assign a different trigger event ID to each stimulus input using the `trigger.blender[N].stimulus[M]` attribute.

The programming example below illustrates how to assign digital I/O events as the stimulus. This example generates a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5:

```
digio.trigger[3].mode = digio.MODE_TRIGGER_OUT
digio.trigger[3].mode = digio.EDGE_RISING
digio.trigger[5].mode = digio.MODE_TRIGGER_OUT
digio.trigger[5].mode = digio.EDGE_RISING
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
```

Action overruns

Action overruns are generated by event blenders depending on the mode, as shown in the following table. Use the status model to monitor for the occurrence of action overruns. For details, see Status model (on page 634).

Mode	Action overrun

And	Generates an overrun when a second event on any of its inputs is detected before generating an output event.
Or	Generates an overrun when two events are detected simultaneously.

Event overruns

When an event occurs, the detector goes into the detected state. Additional events may occur during this time. This sets the overrun state of the detector to `true`, indicating that events have been ignored while the detector was already set. In the case of trigger timers, an overrun is created when a trigger event is detected during the timer delay.

The command interface cannot be overrun.

The following commands can be used to check if a trigger object detector is in an overrun state. These commands return whether the detector has detected more than one event.

Digital I/O	digio.trigger[N].overrun (on page 149)
TSP-Link	tsplink.trigger[N].overrun (on page 325)
Trigger timers	trigger.timer[N].overrun (on page 311)
Event detectors	trigger.detector[N].overrun (on page 305)
Event blenders	trigger.blender[N].overrun (on page 300)

Interactive triggering

The complexity of some test system configurations may not allow a static trigger setup. These configurations require more dynamic control of triggering than the static trigger setup provides. For such cases, a setup providing interactive trigger programming allows the generation and detection of trigger events that can be controlled on demand under remote control.

You can use interactive triggering with digital I/O lines, TSP-Link trigger lines, trigger timers, and trigger generators. This enables the system to generate and detect trigger events on-demand during TSP script execution.

Interactive triggering can provide additional flexibility to the test flow. For example, you can suspend command execution until an event occurs or to create conditional branching to other test setups based on recent measurements.

Generate an interactive trigger

There are two ways to generate an event during operation. The first is to configure the stimulus attribute for an event. When the stimulus is registered, the corresponding event is generated. This includes actions like asserting a trigger on a digital I/O line or starting a timer.

The second method is to manually assert the trigger event. This allows you to generate an event on demand with an automated program or script.

Manually generating a trigger can be combined with trigger model execution using wait blocks. The wait block is configured to use a particular event that is user-generated when you send the assert command. This is useful for applications where you need multiple trigger models to start together, each model starts with a wait block and calling an assert starts all of the models simultaneously.

Be sure to configure the trigger output before sending the assert command. For example, before asserting an output trigger on a particular digital I/O line, first configure its logic and pulse width.

The following commands can be used to generate triggers manually. You cannot assert triggers on a timer.

Digital I/O

digio.trigger[N].assert() (on page 144)	Outputs a trigger on a particular digital I/O line
digio.trigger[N].release() (on page 150)	Releases a line that has been asserted or latched indefinitely

TSP-Link

tsplink.trigger[N].assert() (on page 319)	Outputs a trigger on a particular TSP-Link line
tsplink.trigger[N].release() (on page 326)	Releases a line that has been asserted or latched indefinitely

Event generators

trigger.generator[N].assert() (on page 307)	Generates an event from a particular trigger generator

Detect interactive events

Event detectors monitor for events that match their configuration. They cannot be configured to monitor for events generated by other areas of the instrument. For example, a detector for a digital I/O line that is configured for trigger inputs on rising edges will not detect events from a falling edge.

The detectors run automatically if an object is configured to receive triggers. However, the detectors can also be used to suspend command execution in a script by using the associated `wait()` function. The `wait()` function pauses execution until either the event is detected or the timeout expires. After executing the `wait()` function, the event detector of the trigger object is cleared.

If an event has occurred before the `wait()` function is called, the event detector does not wait for the next event. To force the detector to wait for the next event, clear the detector before issuing the `wait()` command using the associated `clear()` command.

The following commands can be used to wait for a trigger event.

Digital I/O

digio.trigger[N].wait() (on page 154)	Suspends execution until a trigger is detected on an input line or the wait timeout expires
digio.trigger[N].clear() (on page 145)	Clears previous events from the detector

TSP-Link

tsplink.trigger[N].wait() (on page 329)	Suspends execution until a trigger is detected on an input line or the wait timeout expires
tsplink.trigger[N].clear() (on page 320)	Clears previous events from the detector

Trigger timers

trigger.timer[N].wait() (on page 314)	Suspends execution until the trigger timer or the wait timeout expires
trigger.timer[N].clear() (on page 308)	Clears previous events from the detector

Event blenders

trigger.blender[N].wait() (on page 314)	Suspends execution until the event blender detects the events specified by trigger.blender[N].stimulus[M] (on page 302) are detected in the mode specified by the trigger.blender[N].orenable (on page 300) attribute.
trigger.timer[N].clear() (on page 308)	Clears previous events from the detector

Event detectors

trigger.detector[N].wait() (on page 306)	Suspends execution until the event specified by the trigger.detector[N].stimulus (on page 305) attribute is detected
trigger.detector[N].clear() (on page 304)	Clears previous events from the detector

Create a trigger model

This section describes the basic concepts you need to understand to assemble trigger models.

Trigger models are hosted on the module slot they are created for. Multiple channels can be used in a single trigger model, but the channels must exist in the same slot. To use channels across slots, multiple trigger models must be created and synchronized.

NOTE: Before building the trigger model, be sure to configure instrument settings as necessary. This includes configuration lists, sweeps, reading buffers, and events that will be used with the trigger model.

Create a new trigger model

The MP5103 can have multiple named trigger models for each slot. A trigger model must be created and named before blocks can be added. The name must be unique, and it cannot be blank or presently assigned to an existing trigger model.

Using TSP commands:

Send

```
slot[Z].trigger.model.create("triggerModelName")
```

Where

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model to create

Build the trigger model

Once a trigger model is created and named, blocks can be added using remote commands.

Blocks are added to the specified trigger model in the order that the commands are written. Commands do not necessarily have to be sent sequentially, but a new `addblock` command always adds the block to the end of the trigger model. The trigger model runs each block sequentially except when there is a branch block. The action from each block is started and completed before the trigger model moves to the next block. Blocks do not overlap, except for overlapped measurement actions.

Block names can be blank or repeated unless you wish to reference them from a delay or branch block. If so, then you must give it a unique name. Other blocks such as branch blocks can then reference a specific block and return or jump to that block in the model during execution.

The number of blocks in a trigger model depends on available on memory on a module. See Memory considerations for the runtime environment (on page 42) for more information. Within a trigger model, delay and branch blocks can only reference up to 32 different blocks.

Edit a trigger model

Blocks can be removed from the trigger model using remote commands. Blocks after the removed block are not affected, unless another block in the trigger model references the removed block. You cannot edit the trigger model during execution.

Using TSP commands:

Send

```
slot[Z].trigger.model.removeblock("triggerModelName", "triggerBlockName")
```

Where:

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model
- *triggerBlockName* is the name of the trigger block to remove

Delete a trigger model

You can delete a trigger model, which removes all trigger model blocks. To recreate a trigger model, use the slot[Z].trigger.model.create() (on page 616) command.

Using TSP commands:

Send

```
slot[Z].trigger.model.delete("triggerModelName")
```

Where:

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model to delete

Load or unload a trigger model

To minimize latency when executing a trigger model, you can load it to configure both the trigger model and the associated hardware using slot[Z].trigger.model.load() (on page 620). Once loaded, any calls to the slot[Z].trigger.model.initiate() (on page 619) command begin execution more quickly. This can be useful when executing the same test on multiple DUTs, for example. Latency improvements are more apparent when executing larger trigger models with many blocks.

When a trigger model is loaded, the channel is placed into a protected state. The channel will remain in this state until the trigger model is unloaded. Configuration changes that alter the channel's operating mode, source, measure, or range settings are not allowed, though getter and query commands are permitted. Certain contact-check commands are also permitted without unloading the trigger model.

After a trigger model is loaded, the trigger model definition cannot be modified. Commands that add, remove, or edit trigger model blocks are skipped and will generate errors; no changes are made to the trigger model. To make changes to the trigger model, you must unload it.

You do not need to load a trigger model before running it. If you do not load a trigger model, sending the `slot[Z].trigger.model.initiate()` command automatically loads, executes, and then unloads the trigger model upon completion. When a trigger model is loaded with slot[Z].trigger.model.load() (on page 620), the slot[Z].trigger.model.initiate() (on page 619) command skips the load step and runs immediately. The trigger model remains loaded after execution.

When slot[Z].trigger.model.initiate() (on page 619) is called for a previously-loaded trigger model, the mainframe restores all channels used by that trigger model to the pre-load, pre-execution state. This ensures consistent and repeatable operation across multiple runs of the trigger model.

To exit the protected state and allow channel configuration changes or trigger model edits, you can unload the trigger model using slot[Z].trigger.model.unload() (on page 624).

NOTE: A trigger model is automatically unloaded if it is deleted or if a running trigger model is aborted.

Using TSP commands:

Send

```
slot[Z].trigger.model.load("triggerModelName")
```

or

```
slot[Z].trigger.model.unload("triggerModelName")
```

Where:

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model

Run the trigger model

You can run or stop the trigger model by using remote commands.

Multiple trigger models can execute in parallel if they do not use the same channels. For example, if there are two named trigger models that use channel 1 on slot 1, they cannot be run at the same time.

When you run the trigger model, the existing instrument settings are used for any actions unless you assigned configuration lists to the trigger model.

Trigger model operation is an overlapped process. This means that you can run other commands while a trigger model is running if they do not conflict with trigger model operation. For example, you can print the buffer contents, but you cannot change the measure function.

The slot[Z].trigger.model.initiate() (on page 619) command is the overlapped command that starts the process. The command interface is available immediately after the instrument executes the initiate command so that other commands can be executed while the trigger model loads and starts running.

If you change from remote to local control while a trigger model is running, the trigger model is aborted.

Start the trigger model

The slot[Z].trigger.model.initiate() (on page 619) command starts the named trigger model. The trigger model continues to run in the background after this command completes.

To minimize latency between when this command is sent and when the trigger model starts, send the slot[Z].trigger.model.load() (on page 620) command before execution. Refer to Load or unload a trigger model (on page 88) for more information.

Using TSP commands:

Send

```
slot[Z].trigger.model.initiate("triggerModelName")
```

Where:

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model

Abort the trigger model

You can stop a trigger model while it is in progress. When you abort a trigger model, all trigger model commands associated with that named trigger model are terminated.

If the trigger model was loaded using slot[Z].trigger.model.load() (on page 620), it is unloaded during the abort sequence.

Using TSP commands:

Send

```
slot[Z].trigger.model.abort("triggerModelName")
```

Where:

- *Z* is the module slot number
- *triggerModelName* is the name of the trigger model

Check the state of the trigger model

The trigger model can be in one of several states. You can check the status using the slot[Z].trigger.model.state() (on page 623) command.

The following table describes the trigger model states.

TSP command feedback	Description
`trigger.STATE_ABORTED` `trigger.STATE_ABORTING`	Trigger model has been stopped by the user or is in the process of stopping
`trigger.STATE_BUILDING`	The trigger model is created but has not run since new blocks were added
`trigger.STATE_EMPTY`	The trigger model is created but no blocks have been added
`trigger.STATE_FAILED`	The trigger model has stopped because of an error
`trigger.STATE_IDLE`	The trigger model is not running
`trigger.STATE_LOADED`	The trigger model is loaded and has reserved hardware resources for rapid initiates
`trigger.STATE_RUNNING`	The trigger model is actively running

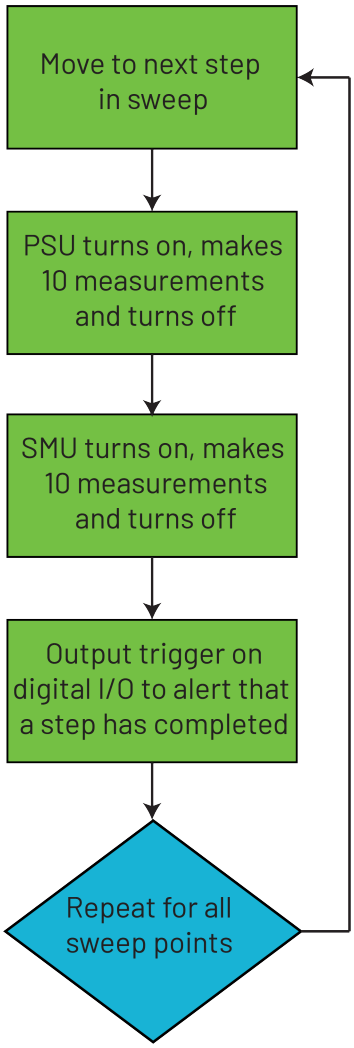
Device settings and performance

Trigger models are optimized for efficiency and speed. During execution, updates to user accessible instrument setting values are suspended. Querying instrument settings while the trigger model is running may return outdated values that do not reflect the actual settings at that moment.

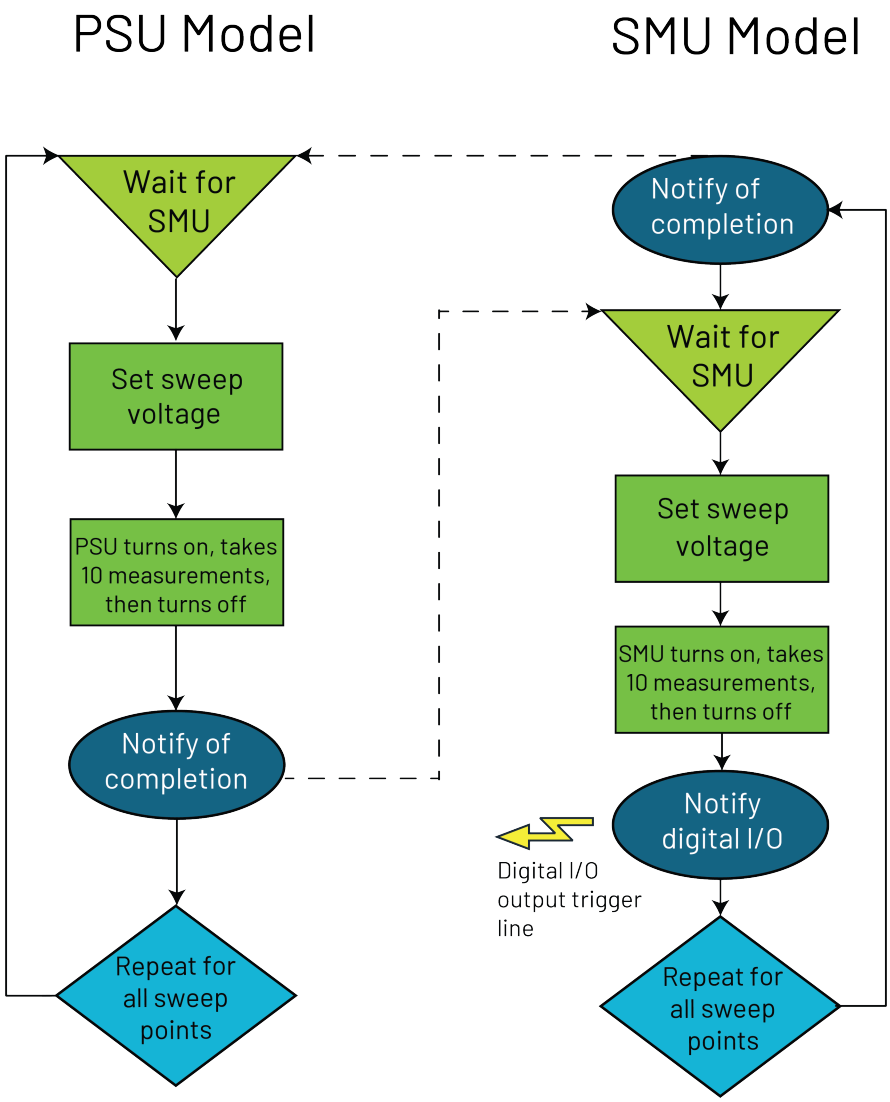
Trigger model programming example

This example demonstrates the construction process for a trigger model. The following graphic references a PSU module in slot 1 and a SMU module in slot 2.

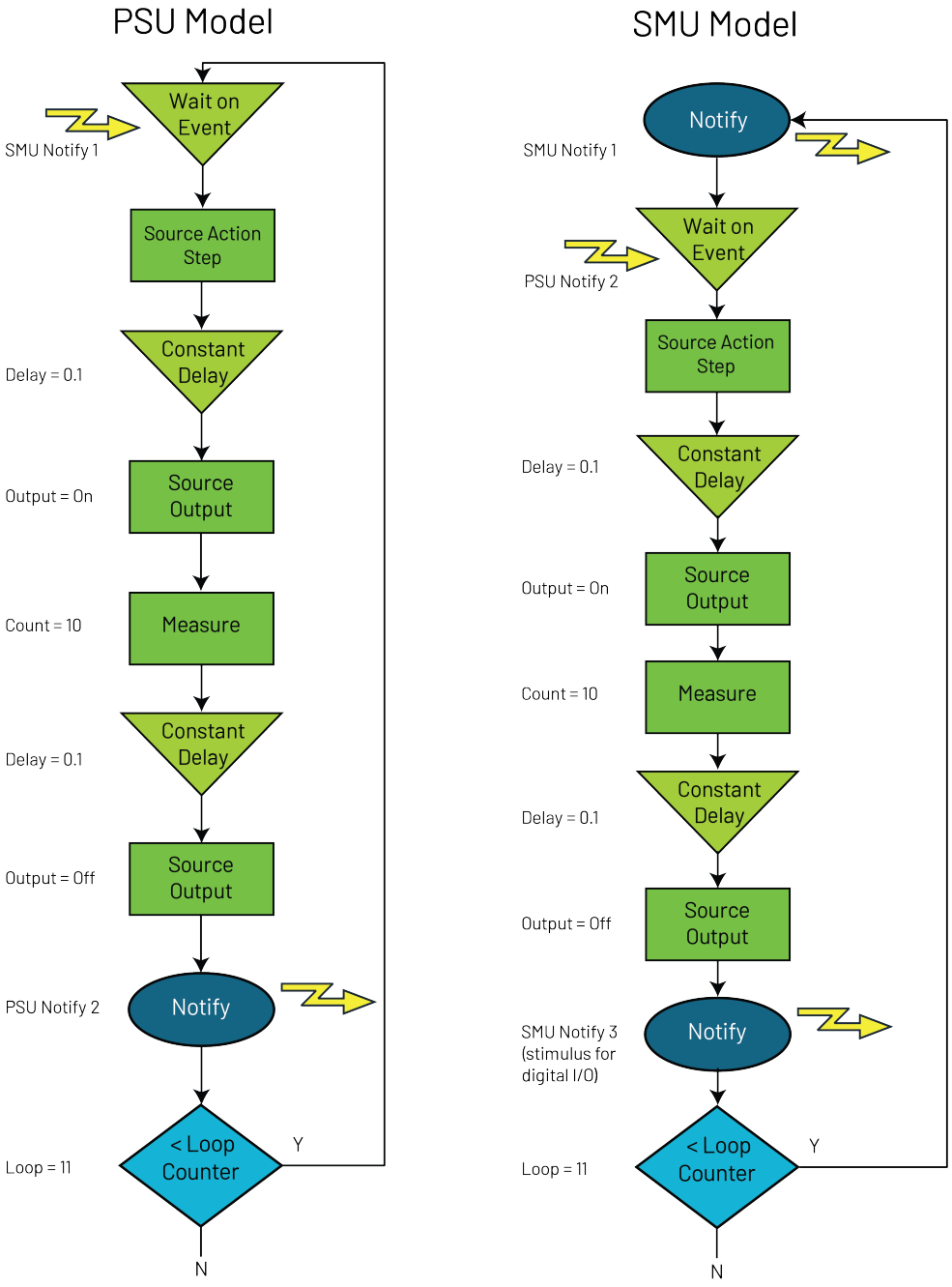
The test flow is shown in the following graphic.



Each module has a trigger model, so this test flow must be split between the actions for the PSU module and the actions for the SMU module. To synchronize the SMU actions to start after the PSU actions have completed, each trigger model must alert the other when an action is completed. Additionally, at the end of the last action, the trigger model must alert the digital I/O line to output a trigger. This expanded test flow is shown in the following figure.



Now that the test flow details the interactions between each model, the action blocks can be broken down into dedicated trigger model blocks that match the actions. The full trigger model flow is shown in the following diagram



This trigger model is now ready to be converted to TSP commands. The following steps must be completed before the trigger model is loaded in the code:

- Configure channel 1 of the PSU to sweep −5 to 5 V
- Assign buffers to measure current and voltage on the PSU
- Configure channel 1 of the SMU to sweep −5 to 5 V
- Assign buffers to measure current and voltage on the SMU
- Configure digital I/O line 1 to be an output trigger line with its stimulus attribute set to the notify event from the SMU trigger model.

The complete set of TSP commands is detailed in the following code blocks.

```
-- Identify channels, slots, and aliases.
local psuSlot = 1
local smuSlot = 2
local smuChan = 1
local psuChan = 1
local smu = slot[smuSlot].smu[smuChan]
local psu = slot[psuSlot].psu[psuChan]
local SMU_TM = slot[smuSlot].trigger.model
local PSU_TM = slot[psuSlot].trigger.model
-- Test settings.
local startV = -5
local stopV = 5
local noPoints = 11
local pi_limit = .5
local ptm_name = "PSU_tm"
local si_limit = .5
local range1 = .1
local stm_name = "SMU_tm"
smu.reset()
psu.reset()
-- Configure the PSU.
psu.source.levelv = startV
psu.source.limiti = pi_limit
-- Set up voltage sweep and reading buffers.
psu.defbuffer1.clear()
psu.defbuffer1.appendmode = 1
psu.defbuffer2.clear()
psu.defbuffer2.appendmode = 1
psu.trigger.source.linearv(startV, stopV, noPoints)
psu.trigger.measure.iv(psu.defbuffer1, psu.defbuffer2)
-- Configure the SMU.
smu.source.func = smu.FUNC_DC_VOLTAGE
smu.source.levelv = startV
smu.source.limiti = si_limit
smu.sense = smu.SENSE_2WIRE
smu.measure.rangei = range1
-- Set up voltage sweep and reading buffers.
smu.defbuffer1.clear()
smu.defbuffer1.appendmode = 1
smu.defbuffer2.clear()
smu.defbuffer2.appendmode = 1
smu.trigger.source.linearv(startV, stopV, noPoints)
smu.trigger.measure.iv(smu.defbuffer1, smu.defbuffer2)
-- Configure digital I/O line 1 to output a trigger when notified.
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
digio.trigger[1].stimulus = SMU_TM.EVENT_NOTIFY3
-- Build the PSU trigger model.
PSU_TM.create(ptm_name)
--Wait for notification from the SMU.
PSU_TM.addblock.wait(ptm_name, "wait-event",SMU_TM.EVENT_NOTIFY1)
-- Advance the voltage sweep to the next step.
PSU_TM.addblock.source.action.step(ptm_name, "linear-advance", psuChan)
-- Delay, turn the output on, make two measurements, and turn output off.
PSU_TM.addblock.delay.constant(ptm_name, "delay1", .1)
PSU_TM.addblock.source.output(ptm_name, "output-on", psuChan, 1)
PSU_TM.addblock.measure(ptm_name, "measure", psuChan, 10)
PSU_TM.addblock.delay.constant(ptm_name, "delay2", .1)
PSU_TM.addblock.source.output(ptm_name, "output-off", psuChan, 0)
-- Notify the SMU of measurement completion.
PSU_TM.addblock.notify(ptm_name, "notify-smu", PSU_TM.EVENT_NOTIFY2)
-- Repeat 11 times through the voltage sweep.
PSU_TM.addblock.branch.counter(ptm_name, "branch-count", "wait-event", noPoints)
-- Build the SMU trigger model.
SMU_TM.create(stm_name)
-- Notify the PSU and wait for a response.
SMU_TM.addblock.notify(stm_name, "notify-psu", SMU_TM.EVENT_NOTIFY1)
SMU_TM.addblock.wait(stm_name, "wait-event", PSU_TM.EVENT_NOTIFY2)
-- Advance the voltage sweep to the next step.
```

```
SMU_TM.addblock.source.action.step(stm_name, "linear-advance", smuChan)
-- Delay, turn the output on, make two measurements, and turn the output off.
SMU_TM.addblock.delay.constant(stm_name, "delay1", .1)
SMU_TM.addblock.source.output(stm_name, "output-on", smuChan, 1)
SMU_TM.addblock.measure(stm_name, "measure", smuChan, 10)
SMU_TM.addblock.delay.constant(stm_name, "delay2", .1)
SMU_TM.addblock.source.output(stm_name, "output-off", smuChan, 0)
-- Send notification for digio to trigger.
SMU_TM.addblock.notify(stm_name, "trigger-digio", SMU_TM.EVENT_NOTIFY3)
-- Repeat 11 times through the voltage sweep.
SMU_TM.addblock.branch.counter(stm_name, "branch-count", "notify-psu", noPoints)
-- Initiate the trigger models and delete them when complete.
PSU_TM.initiate(ptm_name)
SMU_TM.initiate(stm_name)
waitcomplete()
-- Retrieve and display the error log.
print("Events:")
for i = 1, eventlog.count do
    print(eventlog.next())
end
-- Retrieve readings from the PSU, calculate the resistance, and display the results.
print("PSU results:")
print("V\t\t I\t\t R")
for i = 1, psu.defbuffer1.n , 1 do
        print(psu.defbuffer2[i],psu.defbuffer1[i],(psu.defbuffer2[i]/psu.defbuffer1[i]))
end
-- Retrieve readings from the SMU, calculate the resistance, and display the results.
print("\nSMU results:")
print("V\t\t I\t\t R")
for i = 1, smu.defbuffer1.n , 1 do
        print(smu.defbuffer2[i],smu.defbuffer1[i],(smu.defbuffer2[i]/smu.defbuffer1[i]))
end
```

Reading buffers

Introduction to reading buffers

Reading buffers capture measurement and instrument data, including ranges, the output state of the instrument, data collected from trigger models, and instrument status. Reading buffers are stored on the modules.

Each channel for each module in the MP5103 has two default reading buffers. You can also create user-defined reading buffers.

You can perform the following operations on reading buffers:

- Configure reading buffers.
- View reading buffer content.
- Choose to store readings in a default reading buffer or a user-defined reading buffer.
- Save reading buffer content to a USB flash drive.
- Set reading buffers to fill once or fill continuously.
- Change the capacity of user-defined reading buffers.
- Delete user-defined reading buffers. You cannot delete `defbuffer1` and `defbuffer2`.
- Clear reading buffers.

Types of reading buffers

The MP5103 includes default buffers. You can also create additional buffers.

Each module includes two default buffers for each channel. The default buffers are named `defbuffer1` and `defbuffer2`. The size of the default buffers are always 69901.

To access the default buffers, you need to define the slot and channel. For example:

- Slot 1, channel 1: `slot[1].smu[1].defbuffer1.*` or `slot[1].psu[1].defbuffer1.*`
- Slot 1, channel 2: `slot[1].smu[2].defbuffer1.*` or `slot[1].psu[2].defbuffer1.*`
- Slot 2, channel 1: `slot[2].smu[1].defbuffer1.*` or `slot[2].psu[1].defbuffer1.*`

If you define a buffer, you set the slot, channel, and size of the buffer. Refer to Create a user-defined buffer (on page 97) for detail.

Reading buffer contents

The measurements in the reading buffers contain the following elements:

- Reading
- Measure function
- Measure range
- Source function
- Source range
- Source output state (on or off)
- Source value

- Status for the reading, such as if relative offset was applied
- Timestamp, consisting of seconds and fractional seconds
- Base timestamp, consisting of seconds and fractional seconds

Readings start at the first index entry in the empty reading buffer. The relative time is taken from the first reading made after a buffer is cleared. Readings are then made sequentially until the end of the buffer is reached. If the buffer fill mode is set to overwrite old data, readings wrap to the first entry and overwrite the older data. If the fill mode is set to fill once, readings made after the fill count is reached are discarded.

Effects of reset and power cycle on buffers

Reading buffers are not affected by system or module resets (`*RST`, `reset()`, `slot[Z].smu[X].reset()`, or `slot[Z].psu[X].reset()`).

On power cycle, the instrument clears the default buffers and deletes all user-defined buffers.

Create a user-defined buffer

To create a new user-defined reading buffer, you define the variable name and the capacity of the new buffer.

The names for a user-defined buffer must conform to the following rules:

- Start with an alphabetic character.
- Cannot contain periods.
- Cannot exist as a global variable, local variable, or array.

If you create a reading buffer that has the same name as an existing user-defined buffer, the existing buffer is overwritten by the new buffer. Any data in the existing buffer is lost.

There is no fixed limit on the number of user-defined reading buffers you can create. However, you are limited by available memory in the module.

To create a buffer, use one of the following commands:

- For SMUs: slot[Z].smu[X].makebuffer() (on page 493)
- For PSUs: slot[Z].psu[X].makebuffer() (on page 381)

For example, to create a reading buffer named `mybuffer2` with a capacity of 200 readings for channel 2 of a PSU module installed in slot 1, send the following command:

```
mybuffer2 = slot[1].psu[2].makebuffer(200)
```

For a SMU module, send this command:

```
mybuffer2 = slot[1].smu[2].makebuffer(200)
```

Set reading buffer options

You can specify the following settings for the reading buffers:

- **Fill mode:** Determines how the incoming data is managed as the buffer fills.
- **Append mode:** Determines if new data is added to existing buffer data.
- **Cache mode:** Determines how the reading buffer cache is managed.
- **Timestamps:** Determine how timestamps are managed.

Set the fill mode

The fill mode setting for the reading buffer controls how the incoming data is managed as the buffer fills. You can set the read buffer to:

- **Fill once (0):** When the buffer is full, the buffer stops accepting data, no more readings are made, and an error message is displayed.
- **Fill continuously (1):** Data fills the buffer until the end of the buffer or the fill count index is reached. At that point, the data returns to the beginning of the buffer and overwrites the oldest reading.

To set the fill mode, refer to bufferVar.fillmode (on page 353) (PSUs) or bufferVar.fillmode (on page 450) (SMUs).

To set the fill count, refer to bufferVar.fillcount (on page 353) (PSUs) or bufferVar.fillmode (on page 450) (SMUs).

For a buffer that fills once, the first entry starts at index 1 with the timestamp in absolute time. For continuous buffers, the lowest timestamp is after the last entry. For example, if you take 150 readings into a buffer with a capacity of 100, the last reading is at entry 50 and the earliest reading is at 51.

Set the append mode

The append mode determines how new readings are appended to existing measurements in the buffer or if the buffer is cleared before the new readings are stored.

If the append mode is set to `0` (off), any previously stored readings in the buffer are cleared before new ones are stored. If append mode is set to `1`, any previously stored readings remain in the buffer and new readings are added to the buffer following the stored readings.

With append mode set to `1` (on), the first new measurement is stored at reading buffer location `[`*n*`+1]`, where *n* is the number of readings stored in the buffer. When the fill mode is set to once, measurements are only appended to a buffer if the measure count is less than the available space in the buffer. If the buffer cannot hold the readings, an error is returned and no measurements are made.

To set the append mode, refer to bufferVar.appendmode (on page 348) (PSUs) or bufferVar.appendmode (on page 443) (SMUs).

Set the cache mode

The reading buffer cache improves access speed to reading buffer data. However, if you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you initiate successive sweeps but do not reconfigure the sweep measurements. It can also happen when the *bufferVar*`.fillmode` attribute is set to `1`, which may cause reading buffer data to be overwritten.

To prevent the return of stale cache data, disable the cache or make an explicit call to *bufferVar*`.clearcache()`.

To change the cache settings, refer to bufferVar.cachemode (on page 350) (PSUs) or bufferVar.cachemode (on page 446) (SMUs).

To clear stale cache data, refer to bufferVar.clearcache() (on page 352) (PSUs) or bufferVar.clearcache() (on page 448) (SMUs).

Timestamps

Each reading includes a timestamp that is relative to the first measurement that is added to the reading buffer. It is collected at the beginning of the aperture time of each measurement. The finest timestamp resolution is 100e−9 seconds (100 ns). At this resolution, the reading buffer can store unique timestamps for up to 7 minutes. You can increase the resolution for long tests.

The buffer also records the system time for the first reading taken. This time is the base timestamp and can be used to calculate an absolute timestamp for a reading. The base timestamp is saved in number of seconds (whole and fractional) since UTC 12:00 am Jan 1, 1970. The following example converts a relative timestamp to an absolute timestamp for reading 5 in buffer1:

```
baseTimestamp = buffer1.basetimestampseconds + buffer1.basetimestampfractional
```

```
absoluteTimestamp = buffer1.timestamps[5] + baseTimestamp
```

Timestamps are managed using the following commands:

- bufferVar.basetimestampseconds (on page 445) contains the whole seconds portion of the timestamp.
- bufferVar.basetimestampfractional (on page 444) contains the fractional seconds portion of the timestamp.
- bufferVar.timestampresolution (on page 460) sets the resolution of the timestamp.
- bufferVar.timestamps (on page 363) (PSUs) and bufferVar.timestamps (on page 461) (SMUs) contain the relative timestamp for each reading the specified buffer.

Check reading buffer capacity

The capacity of a reading buffer is the number of readings a reading buffer can hold. The capacity of a default buffer is always 69,901. The capacity of a user-defined buffer is set by the size parameter when the buffer is created. To check the capacity of a reading buffer, use the bufferVar.capacity (on page 447) (SMUs) or bufferVar.capacity (on page 351) (PSUs) command.

If a measurement count makes more new measurements than the buffer can hold, the measurement command returns an error and no measurements are made.

The available capacity for reading buffers is affected by other operations of the module. Factors that can affect buffer capacity include:

- Other reading buffers. The total capacity of the reading buffers in the module affects the space that can be allocated to a new buffer.
- Configuration lists.
- Trigger models.
- Sweep configurations.

CAUTION: The MP5103 notifies you when the mainframe runs out of memory. If the mainframe encounters memory allocation errors (errors that specifically state “Out of Memory”), the state of the mainframe cannot be guaranteed. After attempting to save any important data, turn off power to the mainframe and turn it back on to reset the runtime environment and return the mainframe to a known state. Unsaved scripts and reading buffers are lost.

To maximize the memory available for reading buffers:

- Delete user-defined buffers that you are not using. If you power cycle the mainframe, all user-defined buffers in the mainframe and modules are deleted.
- Reduce the number of configuration lists or the number of indexes stored in configuration lists.
- Reduce the number of trigger models or the size of the trigger models. Power cycle the mainframe to remove all loaded trigger models.
- Reduce sweep configuration steps.
- Adjust the `collectgarbage()` settings in Lua. See Introduction to TSP operation (on page 31) for more information.
- Review scripts to improve their memory usage. In particular, you can see memory gains by changing string concatenation lines into a Lua table of string entries. You can then use the `table.concat()` function to create the final string concatenation.

For additional information on memory use by the mainframe, refer to Memory considerations for the runtime environment (on page 42).

Check reading buffer status

Each reading buffer contains a column of status values for each reading. You can return this column using the bufferVar.statuses.[N] (on page 459) attribute.

The returned value for a reading is a binary number where an asserted bit means that a setting or condition was present when the measure was made.

For PSU modules, the following status conditions are available:

- Relative offset was applied to reading
- Current limit was in effect (output voltage was reduced)

For SMU modules, the following status conditions are available:

- 4-wire remote sense mode was enabled
- Relative offset was applied to reading
- Source function limited because the complementary function would be over the set limit

For more information, refer to the command descriptions:

- PSU: bufferVar.statuses (on page 361)
- SMU: bufferVar.statuses (on page 459)

Store data in reading buffers

When you create buffers, the buffers are available to the slot they were created on. They can be used with any command for that slot that takes a buffer parameter. You can also use default slot buffers with commands that take a buffer parameter.

You can get readings by making overlapped or sequential measurements. Overlapped commands do not finish executing before the next command starts. Sequential commands complete execution before the next command starts executing. Overlapped measurements always return readings in a reading buffer. Nonoverlapped measurement functions can return single-point measurement values or store multiple values in a reading buffer. A reading buffer is always used to store measurements during trigger model execution.

Data is stored in the buffers when you make a measurement using the following commands with a buffer defined:

- `slot[Z].psu[Z].measure.Y()`
- `slot[Z].smu[X].measure.Y()`
- `slot[Z].smu[X].measure.overlappedY`
- `slot[Z].psu[X].measure.overlappedY`
- `slot[Z].psu[X].trigger.measure.Y()`
- `slot[Z].smu[X].trigger.measure.Y()`

For example, to measure voltage and store the readings in the `voltMeasBuffer` reading buffer when using channel 1 of a PSU installed in slot 1 of your mainframe, send the command:

```
voltMeasBuffer = slot[1].psu[1].makebuffer(20)
voltMeasBuffer.clear()
slot[1].psu[1].measure.count = 5
slot[1].psu[1].source.levelv = 2.2
slot[1].psu[1].source.output = 1
slot[1].psu[1].measure.v(voltMeasBuffer)
slot[1].psu[1].source.output = 0
```

To print the last reading in the buffer, send the command:

```
print(voltMeasBuffer.readings[voltMeasBuffer.n])
```

Data does not have to be stored in buffers. If you do not provide a buffer name with the measure commands, the channel makes a single reading, which is then returned to Lua when the command is complete.

See the TSP command reference for command descriptions.

Use a reading buffer in a trigger model

Before a trigger model is created and run, you must specify the reading buffer to be used. Use slot[Z].psu[X].trigger.measure.Y() (on page 403) or slot[Z].smu[X].trigger.measure.Y() (on page 530) to assign a reading buffer.

Clear readings from buffers

You can clear all readings and associated columns from a buffer. Associated columns include data such as timestamps and source values.

To explicitly clear the readings from a buffer, use the command *bufferVar*.clear().

Buffers are cleared when you power down or stop a module. In addition, the user-defined buffers are deleted.

If you set *bufferVar*.appendmode to 0 (append mode off), buffers are cleared before new readings are added.

For example, to clear readings from a user-defined buffer named testData, send the command:

```
testData.clear()
```

Save reading buffer content to a USB flash drive

You can save reading buffer content to a USB flash drive by writing to the file, as shown in the following example.

```
-- Insert a flash drive into the front panel of the mainframe.
-- Make 100 measurements.
slot[1].smu[1].measure.count = 100
slot[1].smu[1].measure.v(slot[1].smu[1].defbuffer1)
-- Set file to be created; overwrites if file exists.
local filename = "/usb1/exported_data.csv"
-- Define a file pointer.
local fptr, err = io.open(filename, "w")
-- Check for USB error.
if err == nil then
    fptr:write("Time, Voltage\n")  -- Write the CSC header.
    for i=1, slot[1].smu[1].defbuffer1.n do  -- Iterate through buffer and write data.
        local writeString = string.format("%f, %f\n",
            slot[1].smu[1].defbuffer1.timestamps[i],
            slot[1].smu[1].defbuffer1[i]
        )
        fptr:write(writeString)
    end
-- Flush writing buffer.
    fptr:flush()
-- Close connections to the file.
    fptr:close()
else
    print("Error in Opening USB File")
end
```

Delete a reading buffer

To delete a user-defined reading buffer, set the buffer name to nil and run collectgarbage(). For example, to delete a reading buffer named rb1, send the commands:

```
rb1 = nil
collectgarbage()
```

This example sets the reference to the created reading buffer to nil. If there are no remaining references, collectgarbage() frees the memory, making it available for reuse.

You cannot delete the default buffers defbuffer1 and defbuffer2.

Access the data in buffers

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

You can access individual buffer readings using the `print()` command. For the example above, to return the reading from the 9th measurement, send:

```
print(rb.readings[1])
```

To return multiple types of data from the buffer, you can use the `printbuffer()` command. When you use `printbuffer()`, you define the start and end indexes of the data and the buffer columns to return. You can return multiple columns. Refer to Buffer columns (on page 102) for a list of the available columns.

If you are making simple manual measurements, the returns from the reading buffer are timely and correct. However, if you are using a trigger model or overlapped measurements, some reading buffer columns, such as source values or source ranges, may be stale during execution and updated at completion. When a sweep occurs in a trigger model, some columns, such as returned source values, are not valid.

Buffer columns

The buffer columns are read-only attributes that you can use to access the information contained in an existing buffer. The available columns are:

- bufferVar.measurefunctions (on page 451) (SMUs) or bufferVar.measurefunctions (on page 451) (PSUs): The measurement function that was used to acquire readings stored in a specified reading buffer.
- bufferVar.measureranges (on page 452) (SMUs) or bufferVar.measureranges (on page 452) (PSUs): The measurement range values that were used for readings stored in a specified buffer.
- bufferVar.n (on page 452) (SMUs) or bufferVar.n (on page 355) (PSUs): The number of readings in the buffer.
- bufferVar.readings (on page 453) (SMUs) or bufferVar.readings (on page 356) (PSUs): The acquired readings stored in the specified reading buffer.
- bufferVar.sourcefunctions (on page 454) (SMUs) or bufferVar.sourcefunctions (on page 357) (PSUs): The source function that was used when the readings were stored in a specified reading buffer.
- bufferVar.sourceoutputstates (on page 456) (SMUs) or bufferVar.sourceoutputstates (on page 358) (PSUs): The state of the source output for readings stored in the specified buffer.
- bufferVar.sourceranges (on page 457) (SMUs) or bufferVar.sourceranges (on page 359) (PSUs): The source range used for readings stored in the specified buffer.
- bufferVar.sourcevalues (on page 458) (SMUs) or bufferVar.sourcevalues (on page 360) (PSUs): The source levels that were programmed when the readings in the reading buffer were acquired.
- bufferVar.statuses (on page 459) (SMUs) or bufferVar.statuses (on page 361) (PSUs): The status values of readings in the specified reading buffer.
- bufferVar.timestamps (on page 461) (SMUs) or bufferVar.timestamps (on page 363) (PSUs): The relative timestamp for each reading in the specified buffer.

Refer to the command descriptions for more detail on each of these attributes.

Reading buffer for...do loops

The following examples illustrate the use of `for . . . do` loops to recall data from a reading buffer. The following code examples can be sent as one command line or as part of a script.

Example of printbuffer() with a for...do loop

This example loop uses the `printbuffer()` command to show the readings and relative timestamps for all readings stored in a reading buffer named `buffer1`. The reading and relative timestamps are shown on a single line with the elements comma-delimited.

```
buffer1 = slot[2].smu[1].makebuffer(20)
buffer1.clear()
slot[2].smu[1].measure.count = 5
slot[2].smu[1].source.levelv = 2.2
slot[2].smu[1].source.output = 1
slot[2].smu[1].measure.v(buffer1)
slot[2].smu[1].source.output = 0
```

```
for x = 1, buffer1.n do
     printbuffer(x, x, buffer1.readings, buffer1.timestamps)
end
```

Example comma-delimited output of above code:

```
2.67703e-05, 0.00000e+00
1.74019e-03, 1.66660e-02
1.73940e-03, 3.33320e-02
1.73862e-03, 4.99980e-02
1.73626e-03, 6.66640e-02
```

Example of print() with a for...do loop

The following loop uses the `print()` command instead of the `printbuffer()` command. This loop also returns the readings and relative timestamps for all readings stored in the buffer). However, because the `print()` command is used instead of `printbuffer()`, each line is tab-delimited rather than comma-delimited to produce a columnar output.

```
buffer1 = slot[1].psu[1].makebuffer(20)
buffer1.clear()
slot[1].psu[1].measure.count = 5
slot[1].psu[1].source.levelv = 2.2
slot[1].psu[1].source.output = 1
slot[1].psu[1].measure.v(buffer1)
slot[1].psu[1].source.output = 0
for x = 1, buffer1.n do
     print(buffer1.readings[x], buffer1.timestamps[x])
end
```

Example columnar-delimited output of above code:

```
2.19841e+00     0.00000e+00
2.19838e+00     6.65747e-02
2.19839e+00     1.33151e-01
2.19837e+00     1.99726e-01
2.19834e+00     2.66301e-01
```

Reading buffer command summary

The commands for managing data storage are listed here.

- bufferVar.appendmode (on page 348) (PSU) or bufferVar.appendmode (on page 443) (SMU): Sets the state of the append mode of the reading buffer.
- bufferVar.basetimestampfractional (on page 349) (PSU) or bufferVar.basetimestampfractional (on page 444) (SMU): Contains the fractional seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.
- bufferVar.basetimestampseconds (on page 349) or bufferVar.basetimestampseconds (on page 349): Contains the whole seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.
- bufferVar.cachemode (on page 350) (PSU) or bufferVar.cachemode (on page 446) (SMU): Enables or disables the cache of the specified reading buffer.
- bufferVar.capacity (on page 351) (PSU) or bufferVar.capacity (on page 447) (SMU): Contains the capacity of the reading buffer.
- bufferVar.clear() (on page 351) (PSU) or bufferVar.clear() (on page 447) (SMU): Empties the reading buffer.
- bufferVar.clearcache() (on page 352) (PSU) or bufferVar.clearcache() (on page 448) (SMU): Clears the buffer cache of the specified reading buffer.
- bufferVar.fillcount (on page 354) (PSU) or bufferVar.fillcount (on page 449) (SMU): Sets the number of readings to store before restarting at index 1 when *`bufferVar`*`.fillmode` is set to restart at index 1 when the fill count reaches a specific value.
- bufferVar.fillmode (on page 353) (PSU) or bufferVar.fillmode (on page 450) (SMU): Determines if old data is overwritten when the reading buffer is filled.

- bufferVar.measurefunctions (on page 354) (PSU) or bufferVar.measurefunctions (on page 451) (SMU): Contains the measurement function that was used to acquire readings stored in a specified reading buffer.
- bufferVar.measureranges (on page 355) (PSU) or bufferVar.measureranges (on page 452) (SMU): Contains the measurement range values that were used for readings stored in a specified buffer.
- bufferVar.n (on page 355) (PSU) or bufferVar.n (on page 452) (SMU): Contains the number of readings in the buffer.
- bufferVar.readings (on page 356) (PSU) or bufferVar.readings (on page 453) (SMU): Contains the acquired readings stored in the specified reading buffer.
- bufferVar.sourcefunctions (on page 357) (PSU) or bufferVar.sourcefunctions (on page 454) (SMU): Contains the source function that was used when the readings were stored in a specified reading buffer.
- bufferVar.sourceoutputstates (on page 358) (PSU) or bufferVar.sourceoutputstates (on page 456) (SMU): Contains the state of the source output for readings stored in the specified buffer.
- bufferVar.sourceranges (on page 359) (PSU) or bufferVar.sourceranges (on page 457) (SMU): Contains the source range used for readings stored in the specified buffer.
- bufferVar.sourcevalues (on page 360) (PSU) or bufferVar.sourcevalues (on page 458) (SMUs): Contains the source levels that were programmed when the readings in the reading buffer were acquired.
- bufferVar.statuses (on page 361) (PSU) or bufferVar.statuses (on page 459) (SMU): Contains the status values of readings in the specified reading buffer.
- bufferVar.timestampresolution (on page 362) (PSU) or bufferVar.timestampresolution (on page 460) (SMU): Contains the resolution of the timestamp for each reading stored in a reading buffer.
- bufferVar.timestamps (on page 363) (PSU) or bufferVar.timestamps (on page 461) (SMU): Contains the relative timestamp for each reading in the specified buffer.
- printbuffer() (on page 222): Print data from reading buffer subtables.
- slot[Z].psu[X].defbufferY (on page 380) (PSU) or slot[Z].smu[X].defbufferY (on page 492) (SMU): Contains the default reading buffers for the specific module.
- slot[Z].psu[X].makebuffer() (on page 381) (PSU) or slot[Z].smu[X].makebuffer() (on page 493) (SMU): Creates a user-defined reading buffer for the specific module.

Dual buffer example

The following example shows how to use two default buffers.

```
-- Restore SMU module defaults.
slot[2].smu[1].reset()
-- Select measure I autorange.
slot[2].smu[1].measure.autorangei = slot[2].smu[1].ON
-- Select measure V autorange.
slot[2].smu[1].measure.autorangev = slot[2].smu[1].ON
-- Select ASCII data format.
format.data = format.ASCII
-- Clear buffer 1.
slot[2].smu[1].defbuffer1.clear()
-- Clear buffer 2.
slot[2].smu[1].defbuffer2.clear()
-- Set buffer count to 100.
slot[2].smu[1].measure.count = 100
-- Set measure interval to 0.1 s.
slot[2].smu[1].measure.interval = 0.1
-- Select source voltage function.
slot[2].smu[1].source.func = slot[2].smu[1].FUNC_DC_VOLTAGE
-- Output 1 V.
slot[2].smu[1].source.levelv = 1
-- Turn on output.
slot[2].smu[1].source.output = slot[2].smu[1].ON
-----------
-- Store current readings in buffer 1, voltage readings in buffer 2.
slot[2].smu[1].measure.overlappediv(slot[2].smu[1].defbuffer1, slot[2].smu[1].defbuffer2)
-- Wait for buffer to fill.
waitcomplete()
-- Turn off output.
```

```
slot[2].smu[1].source.output = slot[2].smu[1].OFF
-- Output buffer 1 readings 1 to 100.
print("Readings 1 to 100 from defbuffer1:")
printbuffer(1, 100, slot[2].smu[1].defbuffer1)
-- Output buffer 2 readings 1 to 100.
print("Readings 1 to 100 from defbuffer2:")
printbuffer(1, 100, slot[2].smu[1].defbuffer2)
```

User-defined buffer example

The programming example below illustrates how to store data to a user-defined reading buffer named `MyData1`. The MP5000 Series SMU module stores 100 current readings in `myData1` and then recalls 10 of those readings.

```
-- Restore the SMU module defaults.
slot[2].smu[1].reset()
-- Select measure I autorange.
slot[2].smu[1].measure.autorangei = slot[2].smu[1].ON
-- Select measure V autorange.
slot[2].smu[1].measure.autorangev = slot[2].smu[1].ON
-- Select ASCII data format.
format.data = format.ASCII
--Create buffers named MyData1 that each hold 100 readings.
MyData1 = slot[2].smu[1].makebuffer(100)
MyData1.appendmode = 1
MyData1.fillmode = 1
MyData1.fillcount = 21
--Clear the buffer.
MyData1.clear()
-- Set the measure count to 100.
slot[2].smu[1].measure.count = 100
-- Set the measure interval to 0.1 s.
slot[2].smu[1].measure.interval = 0.1
-- Select the source voltage function.
slot[2].smu[1].source.func = slot[2].smu[1].FUNC_DC_VOLTAGE
-- Set the source voltage to output 1 V.
slot[2].smu[1].source.levelv = 1
-- Turn on output.
slot[2].smu[1].source.output = slot[2].smu[1].ON
-- Store current readings in MyData1.
slot[2].smu[1].measure.overlappedi(MyData1)
-- Wait for the buffer to fill.
waitcomplete()
-- Turn off output.
slot[2].smu[1].source.output = slot[2].smu[1].OFF
-- Output buffer 1 readings 1 to 10.
printbuffer(1, 10, MyData1)
-- Delete MyData1.
MyData1 = nil
collectgarbage()
```

TSP-Link and TSP-Net

Introduction

TSP-Link is a high-speed trigger synchronization and communications bus that test-system builders can use to connect multiple instruments in a master and subordinate configuration. Once connected, all the instruments that are equipped with TSP-Link in a system can be programmed and operated under the control of the master instrument or instruments. This allows the instruments to run tests more quickly because they can be decoupled from frequent computer interaction. The test system can have multiple master and subordinate groups, which can be used to handle multidevice testing in parallel. Combining TSP-Link with a flexible programmable trigger model ensures speed.

Using TSP-Link, multiple instruments are connected and can be used as if they are part of the same physical unit for simultaneous multichannel testing. The test system can be expanded to include up to 32 TSP-Link-enabled instruments.

The TSP-Net™ library allows the MP5000 Series to control LAN-enabled devices directly through its LAN port. This enables the MP5000 Series to communicate directly with a device that is not TSP™ enabled without the use of a controlling computer.

TSP-Link nodes

Each instrument or enclosure attached to the TSP-Link expansion interface is called a node. Each node must be identified with a unique node number. This identification is called a TSP-Link node number.

TSP-Link connections

Connection information is available in the *MP5103 Reference Manual*.

Master and subordinates

In a TSP-Link system, one of the nodes (instruments) is the master node and the other nodes are the subordinate nodes. The master node in a TSP-Link system can control the other nodes (subordinates) in the system. The MP5103 must be the master.

You can use a computer and a remote interface to communicate with a single node in the system. This node becomes the interface to the entire system. When a command is sent through this node, all nodes go into remote operation. The node that receives the command becomes the master and can control all other nodes, which become its subordinates. The master/subordinate relationship between nodes can only be dissolved by performing an abort operation.

Assign node numbers

Each node in the TSP-Link system must have a unique node number. The node number for each instrument is stored in its nonvolatile memory and remains in storage when the instrument is turned off. You can assign node numbers from 1 to 64. However, the system can only include 32 physical nodes.

To assign a node number, for each instrument in the system, set the `tsplink.node` attribute. The command structure is:

```
tsplink.node = N
```

Where *N* = `1` to `64`

Changes to the node number do not take effect until `tsplink.initialize()` is executed on any node in the system.

To determine the node number of an instrument, you can read the `tsplink.node` attribute. Send the following command:

```
print(tsplink.node)
```

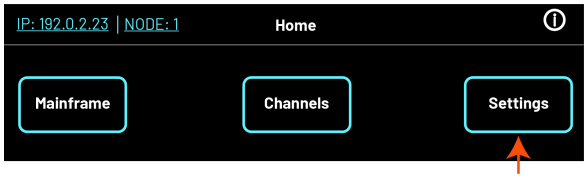
The node number is output.

You can also set the node number for the mainframe using the front panel or web interface.

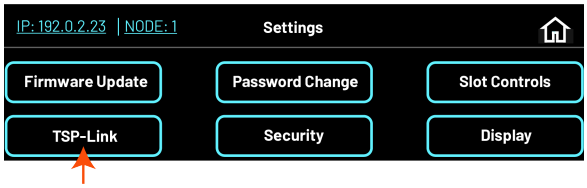
NOTE: From the front panel Home screen, you can also select the `NODE:x` number to open the node setting screen.

To set the mainframe node number from the front panel:

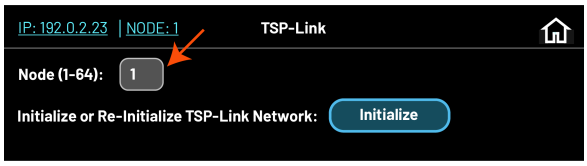
1. From the Home screen, select **Settings**.



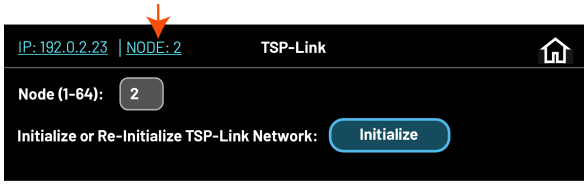
2. Select **TSP-Link**.



3. Select the node field and enter a node number with the on-screen keyboard.



4. The new node number is assigned and is reflected on the network and on other front-panel screens.



To set the mainframe node number from the web interface:

1. From the web home page, select **Settings**.
2. Select **TSP-Link**.
3. Enter a new node number using your keyboard.
4. Press **<Enter>**. The new node number is assigned and is reflected on the TSP-Link network.

Initialize the TSP-Link system

The TSP-Link system must be initialized after configuration changes. You need to initialize the system if you:

- Turn off power or reboot any instrument in the system
- Change node numbers on any instrument in the system
- Rearrange or disconnect the TSP-Link cable connections between instruments

If initialization is not successful, you can check the event log for error messages that indicate the problem. Some typical problems include:

- Two or more instruments in the system have the same node number
- There are no other instruments connected to the instrument that is sending the initialization command
- One or more of the instruments in the system is turned off if the number of expected nodes is defined
- The actual number of nodes is less than the expected number

To initialize the TSP-Link system using remote commands:

Send the command:

```
tsplink.initialize()
```

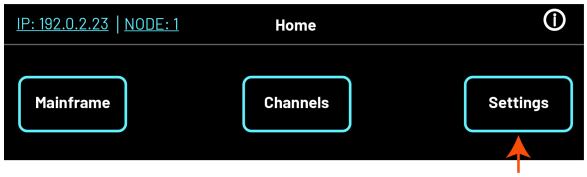
To check the state of the TSP-Link system, send the command:

```
print(tsplink.state)
```

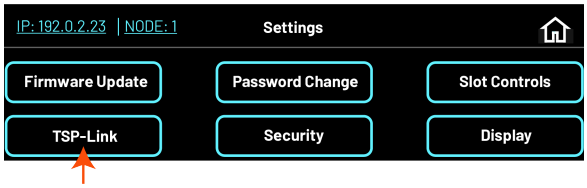
If initialization was successful, `online` is returned. If initialization was not successful, `offline` is returned.

To initialize the TSP-Link system from the front panel of the mainframe:

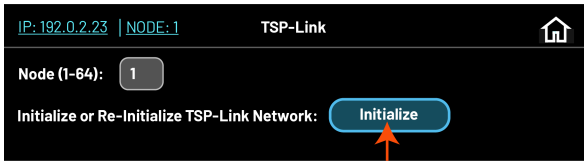
1. From the Home screen, select **Settings**.



2. Select **TSP-Link**.



3. Select **Initialize**.



To initialize the TSP-Link system from the web interface:

1. From the web home page, select **Settings**.
2. Select **TSP-Link**.
3. Select **Initialize**.

Send commands to TSP-Link nodes

Commands for remote nodes are stored in the `node` table. An individual node is accessed as `node[N]`, where *N* is the node number assigned to the node. All TSP-accessible remote commands can be accessed as elements of a specific node.

You can send remote commands to any instrument on the TSP-Link system by adding `node[N].` to the beginning of the remote command, where *N* is the node number. To send a command to the master, you can interact with it as if it were a single instrument.

You do not need to know the node number of the node that is running a script. The variable `localnode` is an alias for the node entry of the node where the script is running. For example, if a script is running on node 5, you can use the global variable `localnode` as an alias for `node[5]`.

For example, to store and recall readings in buffer1 and voltage readings in buffer 2 on an instrument at node 10, you send the following commands:

```
--Set measurement count.
node[10].slot[2].smu[1].measure.count = 100
--Turn on output.
node[10].slot[2].smu[1].source.output = node[10].slot[2].smu.ON
-- Store current readings in buffer 1, voltage readings in buffer 2.
node[10].slot[2].smu[1].measure.overlappediv(node[10].slot[2].smu[1].defbuffer1, node[10].slot[2].s
mu[1].defbuffer2)
-- Wait for buffer to fill and for all TSP-Link nodes.
waitcomplete(0)
--Turn off output.
node[10].slot[2].smu[1].source.output = node[10].slot[2].smu.OFF
-- Output buffer 1 readings 1 to 100.
printbuffer(1, 100, node[10].slot[2].smu[1].defbuffer1)
-- Output buffer 2 readings 1 to 100.
printbuffer(1, 100, node[10].slot[2].smu[1].defbuffer2)
```

Copy test scripts across the TSP-Link network

To run a large script on a remote node, copy the test script to the remote node to increase the speed of test script initiation.

The code in the following example copies a test script across the TSP-Link™ network, creating a copy of the script on the remote node with the same name.

```
-- Add the source code from the script
-- testScript to the data queue.
node[2].dataqueue.add(testScript.source)
-- Create a new script on the remote node
-- using the source code from testScript.
node[2].execute(testScript.name ..
"= script.new(dataqueue.next(), [[" .. testScript.name .. "]])")
```

Resets in a TSP-Link system

From the master node, you can use the `reset()` command to reset all nodes in a TSP-Link system to their default settings.

```
-- Reset all nodes in a TSP-Link system to their default state.
reset()
```

To reset a local group, send:

```
reset(false)
```

To reset a single node, send `node[N].reset()` (where *N* is the node) or `localnode.reset()`. The other nodes are not affected. The following programming example shows this type of reset operation with code that is run on node 1.

```
-- Reset node 1 only.
node[1].reset()
-- Reset node 1 only.
localnode.reset()
-- Reset node 4 only.
node[4].reset().
```

NOTE: Using the `reset()` command in a TSP-Link network differs from using the `tsplink.initialize()` command or the TSP-Link network initialize function from the mainframe front panel or web page interface. The `tsplink.initialize()` command and the front panel and web interfaces reinitialize the TSP-Link network and turns off the output of any TSP-linked instrument; it may also change the state of individual nodes in the system.

Terminate scripts on the TSP-Link system

You can use the `abort` command to terminate a script that is executing on a TSP-Link system. The `abort` command:

- Terminates a script that is executing.
- Returns all nodes to local control.
- Dissolves the master/subordinate relationships between nodes.

Run simultaneous test scripts

You can run test scripts simultaneously to improve functional testing, provide higher throughput, and expand system flexibility. You can use TSP-Link and TSP scripting to run simultaneous test scripts. You can also manage the resources that are allocated to test scripts that are running simultaneously.

In addition, you can use the data queue to do real-time communications between nodes on the TSP-Link system.

To run test scripts simultaneously, you can use groups or run test scripts and programs on remote nodes.

Use groups to manage TSP-Link nodes

TSP-Link groups allow each group to run a different test script simultaneously. This method requires one TSP-Link network and a single remote connection to the computer that is connected to the master node.

A group can consist of one or more nodes. You must assign group numbers to each node using remote commands. If you do not assign a node to a group, it defaults to group 0, which will always be grouped with the master node (regardless of the group to which the master node is assigned).

Group numbers can range from zero (0) to 64. The default group number is 0. You can change the group number at any time. You can also add or remove a node to or from a group at any time.

Each time the power for a node is turned off, the group number for that node changes to 0.

The following example code dynamically assigns a node to a group:

```
--      Assign node 3 to group 1.
node[3].tsplink.group = 1
```

The following table shows an example of the functions of groups on a single TSP-Link network. Each group in this example runs a different test script than the other groups, which allows the system to run multiple tests simultaneously.

TSP-Link network group functions

Group number	Group members	Present function
0	Master node 1	Initiates and runs a test script on node 2. Initiates and runs a test script on node 6. In addition, the master node can execute scripts and process run commands.
1	Group leader node 2	Runs the test script initiated by the master node. Initiates remote operations on node 3 through node 5.
	Node 3 through node 5	Performs remote operations initiated by node 2.

TSP-Link network group functions		
Group number	**Group members**	**Present function**
2	Group leader node 6	Runs the test script initiated by the master node. Initiates remote operations on node 7 through node *n*.
	Node 7 through node *n*	Performs remote operations initiated by node 6.

Master node and groups

You can assign the master node to any group. You can also include other nodes in the group that includes the master. Note that any nodes that are set to group 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

The master node is always the node that coordinates activity on the TSP-Link network.

The master node:

- Is the only node that can use the `execute()` command on a remote node
- Cannot initiate remote operations on any node in a remote group if any node in that remote group is performing an overlapped operation (a command that continues to operate after the command that initiated it has finished running)
- Can execute the `waitcomplete()` command to wait for the group to which the master node belongs; to wait for another group; or to wait for all nodes on the TSP-Link network to complete overlapped operations (overlapped commands allow the execution of subsequent commands while device operations of the overlapped command are still in progress)

Group leader overview

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

The group leader:

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

Run test scripts and programs on remote nodes

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

Use the following code to send the `execute()` command for a remote node. The *N* parameter represents the node number that runs the test script (replace *N* with the node number).

To set the global variable "setpoint" on node N to 2.5:

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

The following code runs a test script that is defined on the local node. For this example, *scriptVar* is defined on the local node, which is the node that initiates the code to run on the remote node. The local node must be the master node.

To run scriptVar on node N:

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

The following code runs a test script that is defined on a remote node. For this example, *scriptVar* is defined on the remote node.

To run a script defined on the remote node:

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

It is recommended that you copy large scripts to a remote node to improve system performance.

Coordinate overlapped operations in remote groups

All overlapped operations on all nodes in a group must complete before the master node can send a command to the group. If you send a command to a node in a remote group when an overlapped operation is running on any node in that group, errors occur.

You can execute the `waitcomplete()` command on the master node or group leader to wait for overlapped operations. The action of `waitcomplete()` depends on the parameters specified.

If you want to wait for completion of overlapped operations for:

- **All nodes in the local group:** Use `waitcomplete()` without a parameter from the master node or group leader.
- **A specific group:** Use `waitcomplete(N)` with a group number as the parameter from the master node. This option is not available for group leaders.
- **All nodes in the system:** Use `waitcomplete(0)` from the master node. This option is not available for group leaders.

For additional information, refer to waitcomplete() (on page 347).

The following code shows two examples that use the `waitcomplete()` command from the master node:

```
-- Wait for each node in group N to complete all overlapped operations.
waitcomplete(N)
-- Wait for all groups on the TSP-Link network to complete overlapped operations.
waitcomplete(0)
```

A group leader can issue the `waitcomplete()` command to wait for the local group to complete all overlapped operations.

The following code is an example of how to use the `waitcomplete()` command from a group leader:

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

Use the data queue for real-time communications

Nodes that are running test scripts at the same time can store data in the data queue for real-time communications. Each instrument has an internal data queue that uses a first-in, first-out (FIFO) structure to store data. You can use the data queue to post numeric values, strings, and tables.

Use the data queue commands to:

- Share data between test scripts running in parallel
- Access data from a remote group or a local node on a TSP-Link network at any time

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

Tables in the data queue consume one entry. When a node stores a table in the data queue, a copy of the data in the table is made. When the data is retrieved from the data queue, a new table is created on the node that is retrieving the data. The new table contains a separate copy of the data in the original table, with no references to the original table or any subtables.

You can access data from the data queue even if a remote group or a node has overlapped operations in process. See the `dataqueue` (on page 138) commands for more information.

Remote TSP-Link commands

Commands that control and access the TSP-Link™ synchronization port are summarized in the following table.

See the MP5000 Series commands (on page 129) for additional details.

TSP-Link commands	
Command	**Description**
tsplink.group (on page 315)	Contains the group number of a TSP-Link node.
tsplink.initialize() (on page 315)	Initializes all nodes in the TSP-Link system.
tsplink.master (on page 316)	Reads the node number assigned to the master node.
tsplink.node (on page 317)	Defines the node number.
tsplink.readbit() (on page 318)	Reads the state of a TSP-Link synchronization line.
tsplink.readport() (on page 318)	Reads the TSP-Link trigger lines as a digital I/O port.
tsplink.state (on page 319)	Describes the TSP-Link online state.
tsplink.trigger[N].assert() (on page 319)	Simulates the occurrence of the trigger and generates the corresponding event ID.
tsplink.trigger[N].clear() (on page 320)	Clears the event detector for a TSP-Link trigger.
tsplink.trigger[N].edge (on page 321)	Configures the edge detection mode for a TSP-Link I/O trigger.
tsplink.trigger[N].EVENT_ID (on page 322)	Constant identifies the number that is used for a trigger event.
tsplink.trigger[N].logic (on page 322)	Sets the output logic of the trigger event generator to positive or negative for the specified line.
tsplink.trigger[N].mode (on page 323)	Defines the trigger operation and detection mode.
tsplink.trigger[N].overrun (on page 325)	Indicates if the event detector ignored an event while in the detected state.
tsplink.trigger[N].pulsewidth (on page 325)	Sets the length of time that the trigger line is asserted for output triggers.
tsplink.trigger[N].release() (on page 326)	Releases a latched trigger on the given TSP-Link trigger line.
tsplink.trigger[N].reset() (on page 326)	Resets some of the TSP-Link trigger attributes to their factory defaults.
tsplink.trigger[N].stimulus (on page 327)	Specifies the event that causes the synchronization line to assert a trigger.
tsplink.trigger[N].wait() (on page 329)	Waits for a trigger.

TSP-Link commands	
Command	**Description**
tsplink.writebit() (on page 330)	Sets a TSP-Link trigger line high or low.
tsplink.writeport() (on page 330)	Writes to all TSP-Link synchronization lines.
tsplink.writeprotect (on page 331)	Contains the write-protect mask that protects bits from changes by the `tsplink.writebit()` and `tsplink.writeport()` functions.

TSP-Link synchronization programming example

The following programming example illustrates how to set bit B1 of the TSP-Link digital I/O port high, and then read the entire port value:

```
tsplink.trigger[1].mode = tsplink.MODE_DIGITAL_OUTPUT
node[10].tsplink.trigger[1].mode = tsplink.MODE_DIGITAL_INPUT
-- Set bit B1 high on output line.
tsplink.writebit(1, 1)
-- Read bit B1 on the input line.
data = node[10].tsplink.readbit(1)
print(data)
```

The output would be similar to:

```
1
```

Use MP5000 TSP-Link commands with other TSP-Link products

If you are connecting the MP5103 in a system with other TSP-Link products, be aware that some of the TSP-Link commands may be different.

Commands that are the same in all TSP-Link products:

- `tsplink.group`
- `tsplink.initialize()`
- `tsplink.master`
- `tsplink.node`
- `tsplink.readport()`
- `tsplink.state`
- `tsplink.writeport()`

MP5103 TSP-Link command	Graphical Series instrument TSP-Link command
`tsplink.readbit()` `tsplink.writebit()`	`tsplink.line[N].state`
`tsplink.trigger[N].assert()`	`trigger.tsplinkout[N].assert()`
`tsplink.trigger[N].clear()`	`trigger.tsplinkin[N].clear()`
`tsplink.trigger[N].edge`	`trigger.tsplinkin[N].edge`
`tsplink.trigger[N].EVENT_ID`	`trigger.EVENT_TSPLINKN`
`tsplink.trigger[N].logic`	`trigger.tsplinkout[N].logic`
`tsplink.trigger[N].mode`	`tsplink.line[N].mode`
`tsplink.trigger[N].overrun`	`trigger.tsplinkin[N].overrun`

MP5103 TSP-Link command	Graphical Series instrument TSP-Link command
`tsplink.trigger[N].pulsewidth`	`trigger.tsplinkout[N].pulsewidth`
`tsplink.trigger[N].release()`	`trigger.tsplinkout[N].release()`
`tsplink.trigger[N].reset()`	`tsplink.line[N].reset()`
`tsplink.trigger[N].stimulus`	`trigger.tsplinkout[N].stimulus`
`tsplink.trigger[N].wait()`	`trigger.tsplinkin[N].wait()`
`tsplink.writeprotect`	Not applicable

Access buffers across TSP-Link nodes

After connecting two TSP-Link™ enabled instruments, you can access buffers over the TSP-Link network.

For local node access to default and user-defined buffers, you do not need a TSP-Link node number.

For user-defined buffers on a remote node, you specify the node number when you create the buffer. After the buffer is created, the buffer name is handled as a local variable, so you do not need the node number to refer to the buffer. You can only use that buffer on the remote node on which it was created.

To use the default buffers on a remote node, you need to use the node number in the command to store readings in the default buffer. You also need the node number to access the default buffer data on a remote node.

The following script example for an SMU module illustrates how and when you need to include a node reference to access default and user-defined buffers when the instrument is part of a TSP-Link network.

```
-- Initialize TSP-Link devices.
tsplink.initialize()
-- Access defbuffer1 on the local node.
slot[1].psu[1].measure.v(slot[1].psu[1].defbuffer1)
-- Access defbuffer1 on a remote node.
node[9].slot[1].smu[1].measure.v(node[9].slot[1].smu[1].defbuffer1)
-- Create and access a user-defined buffer on the local node.
myBuffer = slot[1].psu[1].makebuffer(100)
slot[1].psu[1].measure.v(myBuffer)
-- Access a custom buffer on a remote node.
myRemoteBufferOnNode9 = node[9].slot[1].smu[1].makebuffer(100)
node[9].slot[1].smu[1].measure.v(myRemoteBufferOnNode9)
```

NOTE: It is illegal to reference the user-defined buffer on a different node. For example, `slot[1].psu[1].measure.v(myRemoteBufferOnNode9)` generates an error.

Remove stale values from the reading buffer cache

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

When you run Lua code remotely, the reading buffer data that is held in the cache can become stale. If the values in the reading buffer change while the Lua code runs remotely, another node can hold stale values. Use the *bufferVar*`.clearcache()` command to clear the cache. For additional detail on the reading buffer cache commands, for the PSU, see bufferVar.cachemode (on page 446), bufferVar.cachemode (on page 350) and bufferVar.clearcache() (on page 352); for the SMU, see bufferVar.cachemode (on page 446) and bufferVar.clearcache() (on page 448).

The following example code demonstrates how stale values occur and how to use the `clearcache()` command to clear the cache on node 2, which is part of group 7.

```
-- Create a reading buffer on a node in a remote group.
node[2].tsplink.group = 7
node[2].execute("rbremote = slot[2].smu[1].makebuffer(20)" ..
                "slot[2].smu[1].measure.count = 20  " ..
                "slot[2].smu[1].measure.v(rbremote)")
-- Create a variable on the local node to
```

```
-- access the reading buffer.
rblocal = node[2].getglobal("rbremote")
-- Access data from the reading buffer.
print(rblocal[1])
-- Run code on the remote node that updates the reading buffer.
node[2].execute("slot[2].smu[1]..measure.v(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 never update. Every time the print command is
-- issued after the first print command, the same data buffer
-- values print.
print(rblocal[1])
```

TSP-Net

TSP-Net provides a simple socket-like programming interface to instruments enabled for Test Script Processor (TSP). Using the TSP-Net library, the MP5000 Series can control ethernet-enabled devices directly through its LAN port. This enables the MP5000 Series to communicate directly with a device that is that is not TSP-enabled without the use of a controlling computer.

Using TSP-Net library methods, you can transfer string data to and from a remote instrument, transfer and format data into Lua variables, and clear input buffers. The TSP-Net library is only accessible using commands from a remote command interface when you are using the TSP command language.

While you can use TSP-Net commands to communicate with any ethernet-enabled instrument, specific TSP-Net commands exist for TSP-enabled instruments to allow for support of features unique to the TSP scripting engine. These features include script downloads, reading buffer access, wait completion, and handling of TSP scripting engine prompts.

Using TSP-Net commands with TSP-enabled instruments, a MP5000 Series can download a script to another TSP-enabled instrument and have both instruments run scripts independently. The MP5000 Series can read the data from the remote instrument and either manipulate the data or send the data to a different remote instrument on the LAN.

You can use TSP-Net to connect to a computer. You can use a script on the instrument to transfer data directly to your computer hard drive.

With TSP-Net, you can simultaneously connect to a maximum of 32 devices using standard TCP/IP networking techniques through the LAN port of the MP5000 Series.

Use TSP-Net with any ethernet-enabled instrument

NOTE: Refer to TSP command reference (on page 129) for details about the commands presented in this section.

The MP5000 Series LAN port is autosensing (Auto-MDIX), so you can use either a LAN crossover cable or a LAN straight-through cable to connect directly from the MP5103 to an ethernet device or to a hub.

To set up communications to a remote ethernet-enabled instrument that is enabled for TSP™:

1. Send the following command to configure TSP-Net to send an abort command when a connection to a TSP instrument is established:

   ```
   tspnet.tsp.abortonconnect = 1
   ```

   If the scripts are allowed to run, the connection is made, but the remote instrument may be busy.

2. Send the command:

   ```
   connectionID = tspnet.connect(ipAddress)
   ```

   Where:

   - *connectionID* is the connection ID that is used as a handle in all other TSP-Net function calls.
   - *ipAddress* is the IP address, entered as a string, of the remote instrument.

See tspnet.connect() (on page 332) for additional detail.

To set up communications to a remote ethernet-enabled device that is not enabled for TSP:

Send the command:

```
connectionID = tspnet.connect(ipAddress, portNumber, initString)
```

Where:

- *connectionID* is the connection ID that is used as a handle in all other `tspnet` function calls.
- *ipAddress* is the IP address, entered as a string, of the remote device.
- *portNumber* is the port number of the remote device.
- *initString* is the initialization string to be sent to *ipAddress*.

See tspnet.connect() (on page 332) for additional detail.

To communicate to a remote ethernet device from the MP5103:

1. Connect to the remote device using one of the previous procedures. If the MP5103 cannot make a connection to the remote device, it generates a timeout event. Use `tspnet.timeout` to set the timeout value. The default timeout value is 20 s.
2. Use `tspnet.write()` or `tspnet.execute()` to send strings to a remote device. If you use:
   - `tspnet.write()`: Strings are sent to the device exactly as indicated. You must supply any needed termination characters.
   - `tspnet.execute()`: The instrument appends termination characters to all strings that are sent. Use `tspnet.termination()` to specify the termination character.
3. To retrieve responses from the remote instrument, use `tspnet.read()`. The TSP instrument suspends operation until the remote device responds or a timeout event is generated. To check if data is available from the remote instrument, use `tspnet.readavailable()`.
4. Disconnect from the remote device using the `tspnet.disconnect()` function. Terminate all remote connections using `tspnet.reset()`.

Example script

The following example demonstrates how to connect to a remote device that is not enabled for TSP™, and send and receive data from this device:

```
-- Set tspnet timeout to 5 s.
tspnet.timeout = 5
--      Establish connection to another device with IP address 192.168.1.51
-- at port 1394.
id_instr = tspnet.connect("192.168.1.51", 1394, "*rst\r\n")
-- Print the device ID from connect string.
print("ID is: ", id_instr)
-- Set the termination character to CRLF. You must do this
-- for each connection after the connection has been made.
tspnet.termination(id_instr, tspnet.TERM_CRLF)
-- Send the command string to the connected device.
tspnet.write(id_instr, "login admin\r\n")
-- Read the available data, then print it.
tspnet.write(id_instr, "*idn?\r\n")
print("instrument write/read returns: ", tspnet.read(id_instr))
-- Disconnect all existing TSP-Net sessions.
tspnet.reset()
```

This example produces a return such as:

```
ID is:  1
instrument write/read returns:   SUCCESS: Logged in
instrument write/read returns:   TEKTRONIX,MODEL MP5103,04089762,1.6.3d
```

Remote instrument events

If the MP5000 Series is connected to a TSP-enabled instrument through TSP-Net, all events that occur on the remote instrument are transferred to the event log of the MP5000 Series. The MP5000 Series indicates events from the remote instrument by prefacing these events with “Remote Error.” For example, if the remote instrument generates event code 4909, "Reading buffer not found within device," the MP5000 Series generates the string “Remote Error: (4909) Reading buffer not found within device.”

TSP-Net compared to TSP-Link

The TSP-Link™ network interface is the preferred communications method for most applications when communications need to occur between the MP5000 Series and another TSP-enabled instrument.

One of the advantages of using the TSP-Link network interface is that TSP-Link connections have three trigger lines that are available to each device on the TSP-Link network. You can use any one of the trigger lines to perform hardware triggering between devices on the TSP-Link network. Refer to Synchronous triggering (on page 81).

However, if the distance between the MP5000 Series and the TSP-enabled device is longer than 15 feet, use TSP-Net commands.

TSP-Net instrument commands: General device control

The following instrument commands provide general device control:

TSP-Net instrument commands: TSP-enabled device control

The following instrument commands provide TSP-enabled device control:

tspnet.tsp.abort() (on page 339)
tspnet.tsp.abortonconnect (on page 340)
tspnet.tsp.rbtablecopy() (on page 340)
tspnet.tsp.runscript() (on page 341)

Example: Using tspnet commands

```
function telnetConnect(ipAddress, userName, password)
   -- Connect through telnet to a computer.
   id = tspnet.connect(ipAddress, 23, "")
   -- Read the title and login prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the login name.
   tspnet.write(id, userName .. "\r\n")
   -- Read the login echo and password prompt from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Send the password information.
   tspnet.write(id, password .. "\r\n")
   -- Read the telnet banner from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
end
function test_tspnet()
   tspnet.reset()
   -- Connect to a computer using telnet.
   telnetConnect("192.0.2.1", "my_username", "my_password")
   -- Read the prompt back from the computer.
   print(string.format("from computer--> (%s)", tspnet.read(id, "%n")))
   -- Change directory and read the prompt back from the computer.
   tspnet.write(id, "cd c:\\\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Make a directory and read the prompt back from the computer.
   tspnet.write(id, "mkdir TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- Change to the newly created directory.
   tspnet.write(id, "cd c:\\TEST_TSP\r\n")
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   -- if you have data print it to the file.
   -- 11.2 is an example of data collected.
   cmd = "echo " .. string.format("%g", 11.2) .. " >> datafile.dat\r\n"
   tspnet.write(id, cmd)
   print(string.format("from computer--> (%s)", tspnet.read(id, "%s")))
   tspnet.disconnect(id)
end
test_tspnet()
```

TSP command set reference tables by model

MP5000 Series TSP commands

The following topic contains the TSP command set for the MP5000 Series Modular Precision Test System.

MP5103 mainframe commands

PSU TSP command reference

The following topics contain the TSP command sets for the PSU modules available for the MP5000 Series Modular Precision Test System.

MPSU50-2ST

SMU TSP command reference

The following topics contain the TSP command sets for the SMU modules available for the MP5000 Series Modular Precision Test System.

MSMU60-2

MSMU200-2

Uses all commands available for the MSMU60-2 module.

TSP trigger model commands

The following topics contain the TSP commands used to create, delete, and initiate trigger models for the MP5000 Series Modular Precision Test System and available modules.

Applicable instruments

These commands are used with the following instruments:

- **Mainframe:** MP5103
- **PSU modules:** MPSU50-2ST
- **SMU modules:** MSMU60-2, MSMU200-2

These commands are used with the following instruments:

- **Mainframe:** MP5103
- **SMU modules:** MSMU60-2, MSMU200-2

MP5000 series TSP commands

Introduction

The TSP commands available for the MP5000 modular precision mainframe instrument are listed in alphabetical order.

bit.bitand()

This function performs a bitwise logical AND operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitand(value1, value2)
```

result	Result of the logical AND operation
value1	Operand for the logical AND operation; cannot be a negative number
value2	Operand for the logical AND operation; cannot be a negative number

Details

Any fractional parts of *value1* and *value2* are truncated to form integers. The returned *result* is also an integer.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.bitand(10, 9)
print(testResult)
```

Performs a logical AND operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 8 (binary 1000).

Output:

```
8.00000e+00
```

Also see

Bit manipulation and logic operations (on page 42)
bit.bitor() (on page 130)
bit.bitxor() (on page 130)

bit.bitor()

This function performs a bitwise logical OR operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitor(value1, value2)
```

result	Result of the logical OR operation
value1	Operand for the logical OR operation; cannot be a negative number
value2	Operand for the logical OR operation; cannot be a negative number

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.bitor(10, 9)
print(testResult)
```

Performs a bitwise logical OR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 11 (binary 1011).

Output:

```
1.10000e+01
```

Also see

Bit manipulation and logic operations (on page 42)
bit.bitand() (on page 129)
bit.bitxor() (on page 130)

bit.bitxor()

This function performs a bitwise logical XOR (exclusive OR) operation on two numbers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.bitxor(value1, value2)
```

result	Result of the logical XOR operation
value1	Operand for the logical XOR operation; cannot be a negative number
value2	Operand for the logical XOR operation; cannot be a negative number

Details

Any fractional parts of *value1* and *value2* are truncated to make them integers. The returned *result* is also an integer.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.bitxor(10, 9)
print(testResult)
```

Performs a logical XOR operation on decimal 10 (binary 1010) with decimal 9 (binary 1001), which returns a value of decimal 3 (binary 0011).

Output:

```
3.00000e+00
```

Also see

Bit manipulation and logic operations (on page 42)
bit.bitand() (on page 129)
bit.bitor() (on page 130)

bit.clear()

This function clears a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

result = bit.clear(*value*, *index*)

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to clear: 1 to 32

Details

Any fractional part of *value* is truncated to make it an integer. The returned *result* is also an integer.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.clear(15, 2)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. If you clear the bit at *index* position 2, the returned decimal value is 13 (binary 1101).

Output:

```
1.30000e+01
```

Also see

Bit manipulation and logic operations (on page 42)
bit.get() (on page 132)
bit.set() (on page 134)
bit.test() (on page 136)
bit.toggle() (on page 137)

bit.get()

This function retrieves the weighted value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.get(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get: 1 to 32

Details

This function returns the value of the bit in *value* at *index*. This is the same as returning *value* with all other bits set to zero (0).

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for the number is set to zero (0), the result is zero (0).

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.get(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. If you get the bit at index position 4, the returned decimal value is 8 (binary 1000).

Output:

```
8.00000e+00
```

Also see

Bit manipulation and logic operations (on page 42)
bit.clear() (on page 131)
bit.set() (on page 134)
bit.test() (on page 136)
bit.toggle() (on page 137)

bit.getfield()

This function returns a field of bits from the value starting at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.getfield(value, index, width)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to get: 1 to 32
width	The number of bits to include in the field: 1 to 32

Details

A field of bits is a contiguous group of bits. This function retrieves a field of bits from *value* starting at *index*.

The *index* position is the least significant bit of the retrieved field. The number of bits to return is specified by *width*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
myResult = bit.getfield(13, 2, 3)
print(myResult)
```

The binary equivalent of decimal 13 is 1101.

The field at *index* position 2 and *width* 3 consists of the binary bits 110. The returned value is decimal 6 (binary 110).

Output:

```
6.00000e+00
```

Also see

Bit manipulation and logic operations (on page 42)
bit.get() (on page 132)
bit.set() (on page 134)
bit.setfield() (on page 135)

bit.set()

This function sets a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.set(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to set: 1 to 32

Details

This function returns *result*, which is *value* with the indexed bit set. The *index* must be between 1 and 32.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Any fractional part of *value* is truncated to make it an integer.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.set(8, 3)
print(testResult)
```

The binary equivalent of decimal 8 is 1000. If the bit at *index* position 3 is set to 1, the returned value is decimal 12 (binary 1100).

Output:

```
1.20000e+01
```

Also see

Bit manipulation and logic operations (on page 42)
bit.clear() (on page 131)
bit.get() (on page 132)
bit.getfield() (on page 133)
bit.setfield() (on page 135)
bit.test() (on page 136)
bit.toggle() (on page 137)

bit.setfield()

This function overwrites a bit field at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.setfield(value, index, width, fieldValue)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position in value to set: 1 to 32
width	The number of bits to include in the field: 1 to 32
fieldValue	Value to write to the field

Details

This function returns *result*, which is *value* with a field of bits overwritten, starting at *index*. The *index* specifies the position of the least significant bit of *value*. The *width* bits starting at *index* are set to *fieldValue*.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

Before setting the field of bits, any fractional parts of *value* and *fieldValue* are truncated to form integers.

If *fieldValue* is wider than *width*, the most significant bits of the *fieldValue* that exceed the width are truncated. For example, if *width* is 4 bits and the binary value for *fieldValue* is 11110 (5 bits), the most significant bit of *fieldValue* is truncated and a binary value of 1110 is used.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.setfield(15, 2, 3, 5)
print(testResult)
```

The binary equivalent of decimal 15 is 1111. After overwriting it with a decimal 5 (binary 101) at *index* position 2, the returned `value` is decimal 11 (binary 1011).

Output:

```
1.10000e+01
```

Also see

Bit manipulation and logic operations (on page 42)
bit.get() (on page 132)
bit.set() (on page 134)
bit.getfield() (on page 133)

bit.test()

This function returns the Boolean value (`true` or `false`) of a bit at the specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

result = bit.test(*value*, *index*)

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within value to test: 1 to 32

Details

This function returns *result*, which is the result of the tested bit.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

If the indexed bit for *value* is 0, *result* is `false`. If the bit of *value* at *index* is 1, the returned value is `true`.

If *index* is bigger than the number of bits in *value*, the result is `false`.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.test(10, 4)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Testing the bit at *index* position 4 returns a Boolean value of `true`.

Output:

```
true
```

Also see

Bit manipulation and logic operations (on page 42)
bit.clear() (on page 131)
bit.get() (on page 132)
bit.set() (on page 134)
bit.toggle() (on page 137)

bit.toggle()

This function toggles the value of a bit at a specified index position.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
result = bit.toggle(value, index)
```

result	Result of the bit manipulation
value	Specified number
index	One-based bit position within *value* to toggle: 1 to 32

Details

This function returns *result*, which is the result of toggling the bit *index* in *value*.

Any fractional part of *value* is truncated to make it an integer. The returned value is also an integer.

The indexed bit for *value* is toggled from 0 to 1 or from 1 to 0.

The least significant bit of *value* is at *index* position 1; the most significant bit is at *index* position 32.

The TSP scripting engine stores all numbers internally as IEEE Standard 754 double-precision floating-point values. The logical operations work on 32-bit integers. Fractional bits of any input values are truncated. Returned values are integers. For numbers larger than 4294967295, only the lower 32 bits are used.

Example

```
testResult = bit.toggle(10, 3)
print(testResult)
```

The binary equivalent of decimal 10 is 1010. Toggling the bit at *index* position 3 returns a decimal value of 14 (binary 1110).

Output:

```
1.40000e+01
```

Also see

Bit manipulation and logic operations (on page 42)
bit.clear() (on page 131)
bit.get() (on page 132)
bit.set() (on page 134)
bit.test() (on page 136)

dataqueue.add()

This function adds an entry to the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = dataqueue.add(value)
result = dataqueue.add(value, timeout)
```

result	The resulting value of `true` or `false` based on the success of the function
value	The data item to add; *value* can be of any type
timeout	The maximum number of seconds to wait for space in the data queue

Details

You cannot use the *timeout* value when accessing the data queue from a remote node. You can only use the *timeout* value while adding data to the local data queue.

The *timeout* value is ignored if the data queue is not full.

The `dataqueue.add()` function returns `false`:

- If the timeout expires before space is available in the data queue
- If the data queue is full and a *timeout* value is not specified

If the value is a table, a duplicate of the table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
dataqueue.add(10)
dataqueue.add(11, 2)
result = dataqueue.add(12, 3)
if result == false then
   print("Failed to add 12 to the dataqueue")
end
print("The dataqueue contains:")
while dataqueue.count > 0 do
   print(dataqueue.next())
end
```

Clear the data queue.

Each line adds one item to the data queue.

Output:

```
The dataqueue contains:
1.00000e+01
1.10000e+01
1.20000e+01
```

Also see

dataqueue.CAPACITY (on page 139)
dataqueue.clear() (on page 139)

dataqueue.count (on page 140)
dataqueue.next() (on page 141)
Use the data queue for real-time communications (on page 112)

dataqueue.CAPACITY

This constant is the maximum number of entries that you can store in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
count = dataqueue.CAPACITY
```

count	The variable that is assigned the value of `dataqueue.CAPACITY`

Details

This constant always returns the maximum number of entries that can be stored in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count .. " items in the data queue")
```

This example fills the data queue until it is full and prints the number of items in the queue.

Output:

```
There are 128 items in the data queue
```

Also see

dataqueue.add() (on page 138)
dataqueue.clear() (on page 139)
dataqueue.count (on page 140)
dataqueue.next() (on page 141)
Use the data queue for real-time communications (on page 112)

dataqueue.clear()

This function clears the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
dataqueue.clear()
```

Details

This function forces all `dataqueue.add()` commands that are in progress to time out and deletes all data from the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

```
There are 128 items in the data queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 138)
dataqueue.CAPACITY (on page 139)
dataqueue.count (on page 140)
dataqueue.next() (on page 141)
Use the data queue for real-time communications (on page 112)

dataqueue.count

This attribute contains the number of items in the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

count = dataqueue.count

count	The number of items in the data queue

Details

The count is updated as entries are added with `dataqueue.add()` and read from the data queue with `dataqueue.next()`. It is also updated when the data queue is cleared with `dataqueue.clear()`.

A maximum of `dataqueue.CAPACITY` items can be stored at any one time in the data queue.

Example

```
MaxCount = dataqueue.CAPACITY
while dataqueue.count < MaxCount do
   dataqueue.add(1)
end
print("There are " .. dataqueue.count
   .. " items in the data queue")
dataqueue.clear()
print("There are " .. dataqueue.count
   .. " items in the data queue")
```

This example fills the data queue and prints the number of items in the queue. It then clears the queue and prints the number of items again.

Output:

```
There are 128 items in the data queue
There are 0 items in the data queue
```

Also see

dataqueue.add() (on page 138)
dataqueue.CAPACITY (on page 139)
dataqueue.clear() (on page 139)
dataqueue.next() (on page 141)
Use the data queue for real-time communications (on page 112)

dataqueue.next()

This function removes the next entry from the data queue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = dataqueue.next()
value = dataqueue.next(timeout)
```

value	The next entry in the data queue
timeout	The number of seconds to wait for data in the queue

Details

If the data queue is empty, the function waits up to the *timeout* value.

If data is not available in the data queue before the *timeout* expires, the return value is `nil`.

The entries in the data queue are removed in first-in, first-out (FIFO) order.

If the value is a table, a duplicate of the original table and any subtables is made. The duplicate table does not contain any references to the original table or to any subtables.

Example

```
dataqueue.clear()
for i = 1, 3 do
   dataqueue.add(i)
end
print("There are " .. dataqueue.count .. " items in the data queue")
while dataqueue.count > 0 do
   x = dataqueue.next()
   print(x)
end
print("There are " .. dataqueue.count .. " items in the data queue")
```

Clears the data queue, adds ten entries, then reads the entries from the data queue. Output:

```
There are 3 items in the data queue
1.00000e+00
2.00000e+00
3.00000e+00
There are 0 items in the data queue
```

Your output may differ depending on the setting of `format.asciiprecision`.

Also see

dataqueue.add() (on page 138)
dataqueue.CAPACITY (on page 139)
dataqueue.clear() (on page 139)
dataqueue.count (on page 140)
format.asciiprecision (on page 169)
Use the data queue for real-time communications (on page 112)

delay()

This function delays the execution of the commands that follow it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`delay(`*seconds*`)`

seconds	The number of seconds to delay: 0 s to 100,000 s

Details

The instrument delays execution of the commands for at least the specified number of seconds and fractional seconds. However, the processing time may cause the instrument to delay 5 µs to 10 µs (typical) more than the requested delay.

Example

```
slot.start(1)
delay(10)
slot.stop(1)
```

Powers on the module in slot 1, delays 10 seconds, then powers down the module.

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)
```

data	The present value on the line: 1 or 0
N	Digital I/O line number to be read: 1 to 18

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

```
--Read the value of digital I/O line 2
value = digio.readbit(2)
print("The value of line 2 is: " .. value)
```

Reads the state of digital I/O line 2.

Output

```
The value of line 2 is: 0
```

Also see

digio.readport() (on page 143)
digio.writebit() (on page 155)
digio.writeport() (on page 155)

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()
```

data	The present value of the input lines on the digital I/O port

Details

The binary equivalent of the returned value indicates the value of the input lines on the I/O port. The least significant bit (bit B1) of the binary number corresponds to line 1; bit B18 corresponds to line 18.

For example, a returned value of 170 has a binary equivalent of 000000010101010, which indicates that lines 2, 4, 6, and 8 are high (1), and the other 10 lines are low (0).

Example

```
data = digio.readport()
print(data)
```

Assume lines 2, 4, 6, and 8 are set high when the I/O port is read.

Output:

1.70000e+02

This is binary 10101010.

Also see

digio.readbit() (on page 143)
digio.writebit() (on page 155)
digio.writeport() (on page 155)

digio.trigger[N].assert()

This function asserts a trigger pulse on one of the digital I/O lines

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

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

N	Digital I/O trigger line: 1 to 18

Details

This command asserts a trigger on the specified trigger line. When a trigger is asserted, it generates a pulse with the duration specified by `digio.trigger[N].pulsewidth`.

Example 1

```
digio.trigger[2].assert()
```

Asserts digital I/O trigger line 2.

Example 2

```
digio.trigger[2].logic = digio.LOGIC_NEGATIVE
digio.trigger[2].mode = digio.MODE_TRIGGER_OUT
digio.trigger[2].pulsewidth = 0
digio.trigger[2].assert()
digio.trigger[2].release()
```

Negative trigger out. This example uses indefinite-length triggers. It sets the pulse width to 0 and uses `digio.trigger[N].release()` instead of automatic pulse generation.

Example 3

```
digio.trigger[2].logic = digio.LOGIC_POSITIVE
digio.trigger[2].mode = digio.MODE_TRIGGER_OUT
digio.trigger[2].pulsewidth = 10e-6
digio.trigger[2].assert()
```

Asserts a positive trigger out with a 10e-6 pulse width on digital I/O line 2..

Also see

digio.trigger[N].mode (on page 147)
digio.trigger[N].pulsewidth (on page 150)

digio.trigger[N].clear()

This function clears the 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()
```

N	Digital I/O trigger line: 1 to 18

Details

The event detector of a trigger enters the detected state when an event is detected. It is cleared when `digio.trigger[N].wait()` or `digio.trigger[N].clear()` is called.

`digio.trigger[N].clear()` clears the event detector of the specified trigger line, discards the history of the trigger line, and clears the `digio.trigger[N].overrun` attribute.

Example

```
digio.trigger[2].clear()
```

Clears the trigger event on digital I/O line 2.

Also see

digio.trigger[N].overrun (on page 149)
digio.trigger[N].wait (on page 154)

digio.trigger[N].edge

This command configures the edge detection mode for a digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset digio.trigger[N].reset	Not applicable	digio.EDGE_FALLING

Usage

```
triggerEdge = digio.trigger[N].edge
digio.trigger[N].edge = triggerEdge
```

triggerEdge	The trigger edge selection; refer to **Details** for values
N	Digital I/O trigger line: 1 to 18

Details

This command is used for input trigger detection. To control output trigger generation, use the `digio.trigger[N].logic` command.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Set *triggerEdge* to one of the values shown in the following table.

triggerEdge value	Description
digio.EDGE_FALLING	Detects falling-edge triggers as input when the line is configured as an input or open drain
digio.EDGE_RISING	Detects rising-edge triggers as input when the line is configured as an open drain
digio.EDGE_EITHER	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain

Example

```
digio.trigger[3].edge = digio.EDGE_RISING
```

Sets the trigger edge detection mode for I/O line 3 to digio.EDGE_RISING. This configures line 3 to detect rising edges as input triggers.

Also see

digio.trigger[N].clear() (on page 145)
digio.trigger[N].logic (on page 147)
digio.trigger[N].mode (on page 147)
digio.trigger[N].reset (on page 151)
digio.writeport() (on page 155)

digio.trigger[N].EVENT_ID

This constant 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

eventID	The trigger event number
N	Digital I/O trigger line: 1 to 18

Details

Use this constant to refer to an event generated on trigger line *N*. You can configure the trigger stimulus of another trigger object to respond to this event by setting the stimulus attribute equal to this constant.

Example

```
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.

Also see

digio.trigger[N].stimulus (on page 152)

digio.trigger[N].logic

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset digio.trigger[N].reset	Not applicable	digio.LOGIC_NEGATIVE

Usage

```
triggerLogic = digio.trigger[N].logic
digio.trigger[N].logic = triggerLogic
```

triggerLogic	The trigger mode: ▪ Assert a TTL-high pulse for output: `digio.LOGIC_POSITIVE` ▪ Assert a TTL-low pulse for output: `digio.LOGIC_NEGATIVE`
N	Digital I/O trigger line: 1 to 18

Details

This command is used for output triggers. To control input trigger detection, use the `digio.trigger[N].edge` command.

The digital I/O line must be configured to an applicable trigger mode.

Example

```
digio.trigger[3].mode = digio.MODE_TRIGGER_OUT
digio.trigger[3].logic = digio.LOGIC_POSITIVE
```

Sets I/O line 3 mode to be a trigger output and sets the output logic of the trigger event generator to positive.

Also see

digio.trigger[N].clear() (on page 145)
digio.trigger[N].edge (on page 145)
digio.trigger[N].mode (on page 147)
digio.trigger[N].reset (on page 151)

digio.trigger[N].mode

This attribute sets the mode of the digital I/O line to be a digital line, trigger line, or synchronous line and sets the line to be input, output, or open-drain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset digio.trigger[N].reset	Not applicable	digio.MODE_DIGITAL_IN

Usage

```
triggerMode = digio.trigger[N].mode
digio.trigger[N].mode = triggerMode
```

triggerMode	The trigger mode; see **Details** for values
N	Digital I/O trigger line: 1 to 18

Details

Set *triggerMode* to one of the values shown in the following table.

***triggerMode* value**	**Description**
digio.MODE_DIGITAL_IN	Digital control, input
digio.MODE_DIGITAL_OUT	Digital control, output
digio.MODE_DIGITAL_OPEN_DRAIN	Digital control, open drain
digio.MODE_TRIGGER_IN	Trigger control, input
digio.MODE_TRIGGER_OUT	Trigger control, output
digio.MODE_TRIGGER_OPEN_DRAIN	Trigger control, open drain
digio.MODE_SYNCHRONOUS_MASTER	Synchronous master
digio.MODE_SYNCHRONOUS_ACCEPTOR	Synchronous acceptor

You can use this command to place each digital I/O line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

The following settings of *triggerMode* set the line for direct control as a digital line:

- digio.MODE_DIGITAL_IN: The instrument automatically detects externally generated logic levels. You can read an input line, but you cannot write to it.
- digio.MODE_DIGITAL_OUT: You can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low.
- digio.MODE_DIGITAL_OPEN_DRAIN: Configures the line to be an open-drain signal. The line can serve as an input, an output or both. When a digital I/O line is used as an input in open-drain mode, you must use digio.writebit() or digio.writeport() to write a 1 to it.

The following settings of *triggerMode* set the line for direct control as a trigger line:

- digio.MODE_TRIGGER_IN: The line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This line state uses the edge setting specified by the digio.trigger[*N*].edge attribute.
- digio.MODE_TRIGGER_OUT: The line is automatically set high or low depending on the output logic specified by the digio.trigger[*N*].logic attribute. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
- digio.MODE_TRIGGER_OPEN_DRAIN: Configures the line to be an open-drain signal. You can use the line to detect input triggers or generate output triggers. This line state uses the edge setting specified by the digio.trigger[*N*].edge attribute and the logic setting specified by the digio.trigger[*N*].logic attribute.

When the line is set as a synchronous master, the line detects rising-edge triggers as input. For output, the line asserts a TTL-low pulse.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting an output trigger releases the latched line.

Example

```
digio.trigger[3].mode = digio.MODE_DIGITAL_IN
```

Sets the trigger mode for I/O line 3 to `digio.MODE_DIGITAL_IN`.

Also see

digio.readbit() (on page 143)
digio.trigger[N].edge (on page 145)
digio.trigger[N].EVENT_ID (on page 146)
digio.writebit() (on page 155)

digio.trigger[N].overrun

This function returns the event detector overrun status.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Mainframe reset digio.trigger[N].reset digio.trigger[N].clear()	Not applicable	Not applicable

Usage

overrun = digio.trigger[*N*].overrun

overrun	Trigger overrun state: `true` or `false`
N	Digital I/O trigger line: 1 to 18

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
if overrun then
     print("Trigger overrun set on digital input 1")
else
     print("No trigger overrun on digital input 1")
end
```

Prints the status of the event detector overrun.

Also see

digio.trigger[N].clear() (on page 145)
digio.trigger[N].reset() (on page 151)

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	Power cycle Mainframe reset digio.trigger[N].reset	Not applicable	10e-6 (10 us)

Usage

```
triggerPulseWidth = digio.trigger[N].pulsewidth
digio.trigger[N].pulsewidth = triggerPulseWidth
```

triggerPulseWidth	Pulse width in seconds: 0.0000001 to 400, or 0 for indefinite
N	Digital I/O trigger line: 1 to 18

Details

Setting the pulse width to zero (0) seconds asserts the trigger indefinitely. To release the trigger line use `digio.trigger[N].release()`.

Example

```
digio.trigger[2].pulsewidth = 0.0001
```

Sets the pulse width for I/O trigger line 2 to 100 µs.

Also see

digio.trigger[N].assert() (on page 144)
digio.trigger[N].release() (on page 150)
digio.trigger[N].reset() (on page 151)

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()
```

N	Digital I/O trigger line: 1 to 18

Details

Releases a trigger that was asserted with an indefinite pulsewidth time. It also releases a trigger that was latched in response to receiving a synchronous mode trigger. Only the specified trigger line is affected.

Examples

```
digio.trigger[2].release()
```

Releases digital I/O trigger line 2.

Also see

digio.trigger[N].assert() (on page 144)

digio.trigger[N].pulsewidth (on page 150)

digio.trigger[N].reset()

This function resets digital I/O line values to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
digio.trigger[N].reset()
```

N	Digital I/O trigger line: 1 to 18

Details

This function resets the following attributes to their default values:

- `digio.trigger[N].mode`
- `digio.trigger[N].edge`
- `digio.trigger[N].logic`
- `digio.trigger[N].pulsewidth`
- `digio.trigger[N].stimulus`

It also clears `digio.trigger[N].overrun`.

Example 1

```
digio.trigger[2].reset()
```

Resets the attributes for digital I/O trigger line 2.

Example 2

```
digio.trigger[3].mode = digio.MODE_TRIGGER_OUT
digio.trigger[3].pulsewidth = 50e-6
digio.trigger[3].stimulus = digio.trigger[5].EVENT_ID
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth, digio.trigger[3].stimulus)
digio.trigger[3].reset()
print(digio.trigger[3].mode, digio.trigger[3].pulsewidth, digio.trigger[3].stimulus)
```

Set the digital I/O trigger line 3 for a negative TTL logic pulse with a pulsewidth of 50 µs.

Use digital I/O line 5 to trigger the event on line 3.

Reset the line to factory default values.

Output before reset:

```
4.00000e+00     5.00000e-05     5.00000e+00
```

Output after reset:

```
0.00000e+00     1.00000e-05     0.00000e+00
```

Also see

digio.trigger[N].edge (on page 145)
digio.trigger[N].logic (on page 147)
digio.trigger[N].mode (on page 147)
digio.trigger[N].overrun (on page 149)

digio.trigger[N].pulsewidth (on page 150)
digio.trigger[N].stimulus (on page 152)

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	Power cycle Mainframe reset digio.trigger[N].reset	Not applicable	digio.EVENT_NONE

Usage

```
eventID = digio.trigger[N].stimulus
digio.trigger[N].stimulus = eventID
```

eventID	The event to use as a stimulus; see **Details**
N	Digital I/O trigger line: 1 to 18

Details

Set this attribute to `trigger.EVENT_NONE` to disable the automatic trigger output.

Do not use this attribute to generate output triggers under script control. Use `digio.trigger[N].assert()` instead.

The event IDs that can be used for this command include mainframe-specific event IDs as well as event IDs for any installed modules capable of generating a trigger event. See the following table.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example 1

```
-- Configure digital I/O for positive pulse on stimulus.
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
digio.trigger[1].logic = digio.LOGIC_POSITIVE
digio.trigger[1].stimulus = trigger.timer[1].EVENT_ID
-- Configure a trigger timer to cause digital I/O pulses.
trigger.timer[1].clear()
trigger.timer[1].stimulus = trigger.generator[1].EVENT_ID
trigger.timer[1].count = 10
trigger.timer[1].delay = 100e-3
-- Begin trigger timer.
trigger.generator[1].assert()
```

Set up digital I/O to use a positive pulse and to use a trigger timer.

Example 2

```
-- Alias the module.
local slot_no = 1
local chan_no = 1
local smu = slot[slot_no].smu[chan_no]
-- Configure digital I/O for positive pulse on stimulus.
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
digio.trigger[1].stimulus = slot[slot_no].trigger.model.EVENT_NOTIFY1
digio.trigger[1].logic = digio.LOGIC_POSITIVE
-- Configure the source and measure functions.
smu.source.levelv = 5
smu.source.limiti = 150e-3
smu.trigger.measure.iv(smu.defbuffer1, smu.defbuffer2)
-- Create a trigger model that sources, makes ten measurements, and
-- causes a digital I/O pulse on measurement.
local tm_name = "tm"
local triggerModel = slot[slot_no].trigger.model
triggerModel.delete(tm_name)
triggerModel.create(tm_name)
triggerModel.addblock.source.output(tm_name, "", chan_no, 1)
triggerModel.addblock.measure(tm_name, "meas", chan_no)
triggerModel.addblock.notify(tm_name, "", triggerModel.EVENT_NOTIFY1)
triggerModel.addblock.delay.constant(tm_name, "", 10e-3)
triggerModel.addblock.branch.counter(tm_name,"","meas", 10)
triggerModel.addblock.source.output(tm_name, "", chan_no, 0)
-- Initiate trigger model
triggerModel.initiate(tm_name)
waitcomplete()
print("DONE")
```

Sets up a trigger model to use the digital I/O as a stimulus.

Also see

digio.trigger[N].assert (on page 144)
digio.trigger[N].clear (on page 145)
digio.trigger[N].mode (on page 147)
digio.trigger[N].reset (on page 151)

digio.trigger[N].wait()

This function waits for an input trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = digio.trigger[*N*].wait(*timeout*)

triggered	Whether trigger was detected during the timeout period: ▪ Detected: `true` ▪ No triggers detected: `false`
N	Digital I/O trigger line: 1 to 18
timeout	Timeout in seconds, from 0 to 100000

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 is ready to detect the next trigger. This is true regardless of the number of events detected.

Example

```
triggered = digio.trigger[1].wait(5)
if triggered then
     print("Trigger event detected on pin 1")
else
     print("Timeout waiting for trigger event on pin 1")
end
```

Waits for up to 5 s for a trigger to be detected on trigger line 1, then outputs the results.
Output if no trigger is detected:

```
Timeout waiting for trigger event on pin 1
```

Output if a trigger is detected:

```
Trigger event detected on pin 1
```

Also see

digio.trigger[N].clear (on page 145)
digio.trigger[N].mode (on page 147)

digio.writebit()

This function writes a value to a specific digital I/O line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`digio.writebit(N, value)`

N	Digital I/O trigger line: 1 to 18
value	The value to write to the bit: ▪ Low: 0 ▪ High: Non-zero number

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.

The data must be zero (0) to clear the bit. Any value other than zero (0) sets the bit.

You cannot write to a digital I/O line set to trigger mode or set to input only. The line must be set to a digital control line using `digio.MODE_DIGITAL_OUT` or `digio.MODE_DIGITAL_OPEN_DRAIN`. See `digio.trigger[N].mode` for details.

Example

```
digio.trigger[2].mode = digio.MODE_DIGITAL_OUT
digio.writebit(2, 0)
```

Sets digital I/O line 2 to low (0).

Also see

digio.readbit() (on page 143)
digio.readport() (on page 143)
digio.trigger[N].mode (on page 147)
digio.writeport() (on page 155)
digio.writeprotect (on page 157)

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)`

data	Value to write to the port: 0 to 262143

Details

The binary representation of *data* indicates the output pattern to be written to the I/O port. The least significant bit corresponds to line 1 and the most significant bit corresponds to line 18. For example, a *data* value of 170 has a binary equivalent of 00000010101010. Lines 2, 4, 6, and 8 are set high (1), and the other 10 lines are set low (0). The maximum data value is 262143, which corresponds to all high (1) values.

Write-protected lines are not changed.

The `reset()` function does not affect the present states of the digital I/O lines.

You cannot write to a digital I/O line set to trigger mode or set to input only. The line must be set to a digital control line using `digio.MODE_DIGITAL_OUT` or `digio.MODE_DIGITAL_OPEN_DRAIN`. See `digio.trigger[N].mode` for details.

Example

```
digio.writeport(255)
```

Sets digital I/O lines 1 through 8 high (binary 00000011111111)

Also see

digio.readbit() (on page 143)
digio.readport() (on page 143)
digio.trigger[N].mode (on page 147)
digio.writebit() (on page 155)
digio.writeprotect (on page 157)

digio.writeprotect

This attribute contains 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	Not applicable	Not applicable	0

Usage

```
digio.writeprotect = mask
mask = digio.writeprotect
```

mask	The value that specifies the bit pattern for write-protect: 0 to 262143

Details

Bits that are set to 1 cause the corresponding line to be write-protected.

The binary equivalent of *mask* indicates the mask to be set for the I/O port. For example, a mask value of 7 has a binary equivalent of `00000000000111`. This mask write-protects lines 1, 2, and 3.

The *mask* parameter must be entered as an integer. If a non-integer is entered, it is ignored.

Example

```
writeProtectMask = 15
digio.writeprotect = writeProtectMask
```

Protects digital I/O lines 1, 2, 3, and 4 (binary 00001111).

Also see

digio.writebit() (on page 155)
digio.writeport() (on page 155)

display.screensaver.mode

This attribute selects the screen saver mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
screenmode = display.screensaver.mode
display.screensaver.mode = screenmode
```

screenmode	The screen saver mode: `display.BLANK` (blank screen) or `display.LOGO` (Tektronix logo).

Details

This command selects the screen saver mode.

- Blank screen: `display.BLANK`
- Tektronix logo: `display.LOGO`

Example

```
display.screensaver.mode = display.LOGO
```

Sets the mainframe screen saver to the Tektronix logo.

Also see

display.screensaver.timeout (on page 158)

display.screensaver.timeout

This attribute sets a delay before the mainframe screen saver is activated.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	10

Usage

```
screendelay = display.screensaver.timeout
display.screensaver.timeout = screendelay
```

screendelay	The delay before the mainframe screen saver activates, in minutes: `1`, `10`, or `60`.

Details

This command sets a delay, in minutes, before the screen saver on the front panel of the mainframe is activated.

Example

```
display.screensaver.timeout = 60
```

Sets the delay of the mainframe screen saver to 10 minutes after the last input.

Also see

display.screensaver.mode (on page 158)

eventlog.clear()

This function clears the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventlog.clear()
```

Details

This command removes all events from the event log.

Example

```
eventlog.clear()
```

Clears the event log.

Also see

eventlog.count (on page 159)
eventlog.next() (on page 160)

eventlog.count

This attribute stores the total number of events in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Mainframe reset Clearing event log Reading event log	Not applicable	0

Usage

```
count = eventlog.count
```

count	The total number of events in the event log

Example

```
count = eventlog.count
print(count)
```

Displays the `eventlog` count.

Also see

eventlog.clear() (on page 159)
eventlog.next() (on page 160)

eventlog.next()

This function reads and returns the oldest unread event message from the event log, which is then removed from the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
eventNumber, message, severity, nodeID, timeSeconds, timeFractional = eventlog.next()
```

eventNumber	The event number
message	A description of the event
severity	The severity of the event
nodeID	The TSP-Link node in which the event occurred
timeSeconds	The seconds portion of the time the event occurred
timeFractional	The fractional seconds portion of the time when the event occurred

Details

When an event occurs on the instrument, it is placed in the event log in a first-in, first-out (FIFO) queue. This command retrieves the oldest unread event from the event log and removes it from the event log.

If there are no entries in the event log, the following is returned:

```
0.00000e+00     Queue Is Empty  0.00000e+00     1.00000e+00     0.00000e+00     0.00000e+00
```

To read multiple events, run this command multiple times.

Example

```
print(eventlog.next())
```

Returns the following if no event log entries exist:

```
0.00000e+00     Queue Is Empty  0.00000e+00     1.00000e+00     0.00000e+00     0.
00000e+00
```

Also see

eventlog.clear() (on page 159)
eventlog.count (on page 159)

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()`.

Example

```
print("First Line")
exit()
print("Third Line")
```

The script exits before executing the third line of the code.

Output:

```
First Line
```

Also see

waitcomplete() (on page 347)

fileVar:close()

This function closes the file that is represented by the *fileVar* variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

fileVar`:close()`

fileVar	The file descriptor variable to close

Details

This command is equivalent to `io.close(`*fileVar*`)`.

Note that files are automatically closed when the file descriptors are garbage collected.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
print("\n*** fileVar:close")
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
end
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
myfile:close()
os.remove(fileName)
```

Opens file `myfile.txt` for writing. If no errors were found while opening, writes `Removing file` and closes the file.

Also see

fileVar:flush() (on page 162)
fileVar:read() (on page 163)
fileVar:seek() (on page 165)
fileVar:write() (on page 166)
io.close() (on page 177)
io.open() (on page 180)

fileVar:flush()

This function writes buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

fileVar`:flush()`

fileVar	The file descriptor variable to flush

Details

The *fileVar*`:write()` or `io.write()` functions buffer data, which may not be written immediately to the USB flash drive. Use *fileVar*`:flush()` to flush this data. Using this function removes the need to close a file after writing to it, allowing the file to be left open to write more data. Data may be lost if the file is not closed or flushed before a script ends.

If there is going to be a time delay before more data is written to a file, and you want to keep the file open, flush the file after you write to it to prevent loss of data.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
fileHandle = io.input(fileName)
value = io.read("*a")
print(value)
end
fileHandle:close()
print(eventlog.next())
```

Writes data to a USB flash drive.

Also see

fileVar:write() (on page 166)
io.open() (on page 180)
io.write() (on page 184)

fileVar:read()

This function reads data from a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
data1 = fileVar:read()
data1 = fileVar:read(format1)
data1, data2 = fileVar:read("format1", "format2")
data1, ..., dataN = fileVar:read("format1", ..., "formatN")
```

data1	First data read from the file
data2	Second data read from the file
dataN	Last data read from the file
fileVar	The descriptor of the file to be read
format1	A string or number indicating the first type of data to be read
format2	A string or number indicating the second type of data to be read
formatN	A string or number indicating the last type of data to be read

`...`	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

Format parameter	Description
`"*n"`	Returns a number
`"*a"`	Returns the whole file, starting at the present position; returns an empty string if it is at the end of file
`"*l"`	Default setting; returns the next line and skips the end of line character; returns `nil` if the present file position is at the end of file
N	Returns a string with the number of characters specified by *N*: ▪ If *N* is 5, 5 characters are returned ▪ If fewer than *N* characters are available, all characters are returned ▪ If *N* is zero (0), returns an empty string ▪ If the present file position is at the end of file, returns `nil`

If no format parameters are provided, the function performs as if the function is passed the value `"*l"`.

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
print("fileVar:read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
contents = myfile:read("*a")
print(contents)
end
myfile:close()
os.remove(fileName)
```

Reads data from the input file.

Also see

fileVar:write() (on page 166)
io.input() (on page 178)
io.open() (on page 180)

fileVar:seek()

This function sets and gets the present position of a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
position, errorMsg = fileVar:seek()
position, errorMsg = fileVar:seek("whence")
position, errorMsg = fileVar:seek("whence", offset)
```

position	The new file position, measured in bytes from the beginning of the file
errorMsg	A string containing the error message
fileVar	The file descriptor variable
whence	A string indicating the base against which *offset* is applied; the default is `"cur"`
offset	The intended new position, measured in bytes from a base indicated by *whence* (default is `0`); the offset can be positive or negative depending on which direction to move the new file position.

Details

The *whence* parameters may be any of the following:

- Beginning of file: `"set"`
- Current position: `"cur"`
- End of file: `"end"`

If an error is encountered, it is logged to the event log, and the command returns `nil` and the error string.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** fileVar:seek")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
position = myfile:seek("end", -1)
print(position)
end
myfile:close()
os.remove(fileName)
```

Get the present position of a file.

Also see

io.open() (on page 180)

fileVar:write()

This function writes data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar:write(data)
fileVar:write(data1, data2)
fileVar:write(data1, ..., datan)
```

fileVar	The file descriptor variable
data	Write all data to the file
data1	The first data to write to the file
data2	The second data to write to the file
datan	The last data to write to the file
...	One or more entries (or values) separated by commas

Details

This function may buffer data until a flush (*fileVar*:flush() or io.flush()) or close (*fileVar*:close() or io.close()) operation is performed.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** fileVar:write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
do
myfile:write("Line 1")
end
myfile:close()
os.remove(fileName)
```

Write data to a file.

Also see

fileVar:close() (on page 161)

fileVar:flush() (on page 162)
io.close() (on page 177)
io.flush() (on page 177)
io.open() (on page 180)

firmware.image.load()

This function loads a firmware image file from a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
firmware.image.load("/usb1/filename")
```

filename	Name of the firmware file to load, including the directory if used

Details

After loading the image, it can be installed on the mainframe or module using `firmware.update()` or `slot[Z].firmware.update()`.

CAUTION: Do not turn off power until the update process is complete.

This command replaces `firmware.load()`, which is deprecated.

Example 1

```
firmware.image.load("/usb1/MP5000-1.0.0.x")
firmware.update()
```

Load the mainframe firmware file `mp5000-v1.0.0.x` on the USB drive inserted into the front of the mainframe.

Update the firmware.

Example 2

```
firmware.image.load("/usb1/module/MPSU50-2ST-1.0.0.x")
slot[1].firmware.update()
```

Load the module firmware `MPSU50-2ST-1.0.0.x` file in the `module` folder of the USB drive inserted into the front of the mainframe.

Update the module in slot 1.

Also see

"Update the firmware using TSP" in the *MP5000 Reference Manual*
firmware.update() (on page 168)
slot[Z].firmware.update() (on page 463)

firmware.image.valid

This attribute indicates whether or not the loaded firmware is valid.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle	Not applicable	False

Usage

```
valid = firmware.image.valid
```

valid	Status of the firmware validation: ▪ Firmware image is valid: `true` ▪ Firmware image is corrupted or missing: `false`

Details

This attribute indicates if the loaded firmware image has been successfully validated. If the image has been validated and can be used to update the mainframe or a module, this attribute will be `true`. If the image is corrupted or missing, this attribute will be `false`.

Example

```
print(firmware.image.valid)
```

Might return

```
true
```

Indicating that the loaded firmware image is valid.

Also see

firmware.image.load() (on page 167)

firmware.update()

This function updates the firmware of the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
firmware.update()
```

Details

This command updates the mainframe firmware and restarts the instrument.

Before issuing this command, you need to load the firmware file onto the instrument using `firmware.image.load()`.

CAUTION: Do not turn off power until the update process is complete.

Example

```
firmware.image.load("/usb1/MP5000-1.0.0.x")
firmware.update()
```

Load the mainframe firmware file `mp5100-v1.0.0.x` on the USB drive inserted into the front of the mainframe.

Update the firmware.

Also see

"Update the firmware using TSP" in the *MP5000 Reference Manual*
firmware.image.load() (on page 167)
slot[Z].firmware.update() (on page 463)

format.asciiprecision

This attribute sets the precision (number of digits) for all numbers returned in the ASCII format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle Mainframe reset	Not saved	6

Usage

```
precision = format.asciiprecision
format.asciiprecision = precision
```

precision	A number representing the number of digits to be printed for numbers printed with the `print()`, `printbuffer()`, and `printnumber()` functions: 1 to 16; set to 0 to use the native Lua format

Details

This attribute specifies the precision (number of digits) for numeric data printed with the `print()`, `printbuffer()`, and `printnumber()` functions. The `format.asciiprecision` attribute is only used with the ASCII format.

Note that the precision is the number of significant digits printed. There is always one digit to the left of the decimal point; be sure to include this digit when setting the precision.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x)
format.asciiprecision = 0
printnumber(x)
```

Output:

```
2.540000000e+00
2.54e+00
2.540000e+00
```

Also see

format.byteorder (on page 169)
format.data (on page 170)
print() (on page 221)
printbuffer() (on page 222)
printnumber() (on page 225)

format.byteorder

This attribute sets the binary byte order for the data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset	Not saved	format.LITTLEENDIAN

Usage

```
order = format.byteorder
format.byteorder = order
```

order	Byte order value as follows: ▪ Most significant byte first: `format.BIGENDIAN`, `format.NORMAL`, `format.NETWORK` ▪ Least significant byte first: `format.LITTLEENDIAN` or `format.SWAPPED`

Details

This attribute selects the byte order in which data is written when you are printing data values with the `printnumber()` and `printbuffer()` functions. The byte order attribute is only used with the `format.data` settings `format.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64`.

`format.NORMAL`, `format.BIGENDIAN`, and `format.NETWORK` select the same byte order. `format.SWAPPED` and `format.LITTLEENDIAN` select the same byte order. Selecting which to use is a matter of preference.

Select the `format.SWAPPED` or `format.LITTLEENDIAN` byte order when sending data to a computer with a Microsoft Windows operating system.

Example

```
x = 1.23
format.data = format.REAL32
format.byteorder = format.LITTLEENDIAN
printnumber(x)
format.byteorder = format.BIGENDIAN
printnumber(x)
```

The output depends on the terminal program you use, but it looks something like:

```
#0¤p??
#0??p¤
```

Also see

format.asciiprecision (on page 169)
format.data (on page 170)
printbuffer() (on page 222)
printnumber() (on page 225)

format.data

This attribute sets the data format for data that is printed using the `printnumber()` and `printbuffer()` functions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Power cycle Mainframe reset	Not saved	format.ASCII

Usage

```
value = format.data
format.data = value
```

value	The format to use for data, set to one of the following values: ▪ ASCII format: `format.ASCII` ▪ Single-precision IEEE Std 754 binary format: `format.REAL32` or `format.SREAL` ▪ Double-precision IEEE Std 754 binary format: `format.REAL64`, `format.REAL`, or `format.DREAL`

Details

The precision of numeric values can be controlled with the `format.asciiprecision` attribute. The byte order of `format.SREAL`, `format.REAL`, `format.REAL32`, and `format.REAL64` can be selected with the `format.byteorder` attribute.

REAL32 and SREAL select the same single precision format. REAL and REAL64 select the same double-precision format. They are alternative identifiers. Selecting which to use is a matter of preference.

The IEEE Std 754 binary formats use four bytes for single-precision values and eight bytes for double-precision values.

When data is written with any of the binary formats, the response message starts with #0 and ends with a new line. When data is written with the ASCII format, elements are separated with a comma and space.

NOTE: Binary formats are not intended to be interpreted by humans.

Example

```
format.asciiprecision = 10
x = 3.14159265
format.data = format.ASCII
printnumber(x)
format.data = format.REAL64
printnumber(x)
```

Output a number represented by `x` in ASCII using a precision of 10, then output the same number in binary using double-precision format.

```
Output:
3.141592650e+00
#0ñÔÈSû!   @
```

Also see

format.asciiprecision (on page 169)
format.byteorder (on page 169)
printbuffer() (on page 222)
printnumber() (on page 225)

fs.chdir()

This function sets the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`fs.chdir("path")`

path	A string indicating the new working directory path

Details

The new working directory path may be absolute or relative to the current working directory.

An error is logged to the event log if the given path does not exist.

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.

Verify that `/usb1/temp` is a directory and change it to be the current working directory.

Set the variable for the current working directory to be `testPath`.

The return should be:

`/usb1/temp`

If `/usb1/temp` is not a directory, set the variable for the current working directory to be `testPath`.

The return is:

`/usb1`

Also see

None

fs.cwd()

This function returns the absolute path of the current working directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

path = fs.cwd()

path	The absolute path of the current working directory

Example

```
if fs.is_dir("/usb1/temp") == true then
  fs.chdir("/usb1/temp")
  testPath = fs.cwd()
  print(testPath)
else
  testPath = fs.cwd()
  print(testPath)
end
```

Insert a USB flash drive into the front panel of the instrument.

Verify that `/usb1/temp` is a directory and change it to be the current working directory.

Set the variable for the current working directory to be `testPath`.

The return should be:

`/usb1/temp`

If `/usb1/temp` is not a directory, set the variable for the current working directory to be `testPath`.

The return is:

`/usb1`

Also see

None

fs.is_dir()

This function tests whether or not the specified path refers to a directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

status = fs.is_dir("*path*")

status	Whether or not the given path is a directory: `true` or `false`
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example 1

```
print("Is directory: ", fs.is_dir("/usb1/"))
```

Because `/usb1/` is always the root directory of an inserted flash drive, you can use this command to verify that USB flash drive is inserted.

Example 2

```
if fs.is_dir("/usb1/temp") == false
and fs.is_file("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.

Check to see if the `temp` directory exists.

If it does not exist, create a directory named `temp`.

Also see

fs.is_file() (on page 174)

fs.is_file()

Tests whether the specified path refers to a file (as opposed to a directory).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status = fs.is_file("path")
```

status	The state of the specified path: ▪ If the given path is a file: `true` ▪ If the given path is not a file: `false`
path	The path of the file system entry to test

Details

The file system path may be absolute or relative to the current working system path.

Example

```
rootDirectory = "/usb1/"
print("Is file: ", fs.is_file(rootDirectory))
```

Insert a USB flash drive into the front panel of the instrument.

Set `rootDirectory` to be the USB port.

Check to see if `rootDirectory` is a file. Because `rootDirectory` is the path to a directory, the return is `false`.

Also see

fs.is_dir() (on page 173)

fs.mkdir()

This function creates a directory or file at the specified path.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
path = fs.mkdir("newPath")
```

path	The returned path of the new directory
newPath	Location (path) of where to create the new directory or file

Details

The directory path may be absolute or relative to the current working directory.

An error is logged to the event log if the parent folder of the new directory does not exist, or if a file system entry already exists at the given path.

Example

```
if fs.is_dir("/usb1/temp") == false
and fs.is_file("/usb1/temp") == false then
    fs.mkdir("/usb1/temp")
end
```

Insert a USB flash drive into the front panel of the instrument.

Check to see if the `temp` directory exists.

If it does not exist, create a directory named `temp`.

Also see

fs.rmdir() (on page 176)

fs.readdir()

This function returns a list of the file system entries in the directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

files = fs.readdir("*path*")

files	A table containing the names of all the file system entries in the specified directory
path	The directory path

Details

The directory path may be absolute or relative to the current working directory.

This command is nonrecursive. For example, entries in subfolders are not returned.

An error is logged to the error queue if the given path does not exist or does not represent a directory.

Example

```
rootDirectory = "/usb1/"
entries = fs.readdir(rootDirectory)
count = table.getn(entries)
print("Found a total of "..count.." files and directories")
for i = 1, count do
   print(entries[i])
end
```

Insert a USB flash drive into the front panel of the instrument.

Set `rootDirectory` to be the USB port.

Set `entries` as the variable for the file system entries in `rootDirectory`.

Return the number of files and directories in the directory.

Also see

None

fs.rmdir()

This function removes directories and files from the file system.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
fs.rmdir("path")
```

path	The path of the directories and files to remove

Details

This path may be absolute or relative to the current working directory.

An error is logged to the event log if the given path does not exist or does not represent a directory. An error is also logged if the directory is not empty.

Example

```
rootDirectory = "/usb1/"
tempDirectoryName = "temp"
if fs.is_dir(rootDirectory..tempDirectoryName) == false then
    fs.mkdir(rootDirectory..tempDirectoryName)
end
fs.rmdir(rootDirectory..tempDirectoryName)
```

Insert a USB flash drive into the front panel of the instrument.

Set `rootDirectory` to be the USB port.

Set `tempDirectoryName` to be equivalent to `temp`.

Check to see if `tempDirectoryName` exists.

If it does not exist, create a directory named `temp`.

Remove the directories and files.

Also see

fs.mkdir() (on page 174)

gettimezone()

This function retrieves the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timeZone = gettimezone()
```

timeZone	The local time zone of the instrument

Details

See `settimezone()` for additional details about the time zone format and a description of the fields.

timeZone can be in either of the following formats:

- If one parameter was used with `settimezone()`, the format used is:
  `GMThh:mm:ss`
- If four parameters were used with `settimezone()`, the format used is:
  `GMThh:mm:ssGMThh:mm:ss,Mmm.w.dw/hh:mm:ss,Mmm.w.dw/hh:mm:ss`

Example

```
timezone = gettimezone()
```

Reads the value of the local time zone.

Also see

settimezone() (on page 234)

io.close()

This function closes a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
io.close()
io.close(file)
```

file	The descriptor of the file to close

Details

If a file is not specified, the default output file closes.

Only `io.close()`, used without specifying a parameter, can be accessed from a remote node.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if nil == testError then
   testFile:write("This is my test file")
   io.close(testFile)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `"This is my test file"` and closes the file.

Also see

io.open() (on page 180)

io.flush()

This function saves buffered data to a file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
io.flush()
```

Details

You must use the `io.flush()` or `io.close()` functions to write data to the file system.

NOTE: Data is not automatically written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

This function only flushes the default output file.

Using this command removes the need to close a file after writing to it and allows it to be left open to write more data. Data may be lost if the file is not closed or flushed before an application ends. To prevent the loss of data if there is going to be a time delay before more data is written (and when you want to keep the file open and not close it), flush the file after writing to it.

Also see

fileVar:flush() (on page 162)
fileVar:write() (on page 166)
io.write() (on page 184)

io.input()

This function assigns a previously opened file, or opens a new file, as the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.input()
fileVar = io.input("newfile")
```

fileVar	The descriptor of the input file or an error message (if the function fails)
newfile	A string representing the path of a file to open as the default input file, or the file descriptor of an open file to use as the default input file

Details

The *newfile* path may be absolute or relative to the current working directory.

When using this function from a remote TSP-Link™ node, this command does not accept a file descriptor and does not return a value.

Example

```
local fileName = "/usb1/test.txt"
-- Open a file in read mode.
myfile, myfile_err, myfile_errnum = io.open(fileName, "r")
-- Check for errors.
if myfile_err == nil then
    -- Close the file.
    myfile.close()
    -- Read and print all lines of file.
    local line = ""
    -- Set the file to the default input file.
    fileHandle = io.input(fileName)
    while line ~= nil do
        line = io.read()
        print(line)
    end
    io.close(fileHandle)
else
    print("Error in opening file")
end
```

Opens file `testfile.txt` for reading. If no errors were found while opening, reads and prints all lines of the file.

Also see

io.open() (on page 180)
io.output() (on page 181)

io.open()

This function opens a file for later reference.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
fileVar, errorMsg = io.open("path")
fileVar, errorMsg = io.open("path", "mode")
```

fileVar	The descriptor of the opened file
errorMsg	Indicates whether an error was encountered while processing the function
path	The path of the file to open
mode	A string representing the intended access mode; see **Details**: ▪ Read mode: `"r"` ▪ Write mode: `"w"` ▪ Append mode: `"a"`

Details

The path to the file to open may be absolute or relative to the current working directory. If you successfully open the file, *errorMsg* is nil and *fileVar* has the descriptor used to access the file.

The access mode can be:

- Read mode: Opens the file as read-only. The file cannot be edited.
- Write mode: Opens the file with the file position at the beginning. This overwrites existing content.
- Append mode: Opens the file with the file position at the end. This adds to existing content.

A file must be closed and reopened to change access modes.

If an error is encountered, the command returns `nil` for *fileVar* and an error string.

Example

```
testFile, testError = io.open("testfile.txt", "w")
if testError == nil then
   testFile:write("This is my test file")
   io.close(testFile)
end
```

Opens file `testfile.txt` for writing. If no errors were found while opening, writes `This is my test file` and closes the file.

Also see

io.close() (on page 177)
os.remove() (on page 219)

io.output()

This function assigns a previously opened file or opens a new file as the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see **Details**)			

Usage

```
fileVar = io.output()
fileVar = io.output("newfile")
```

fileVar	The descriptor of the output file or an error message (if the function fails)
newfile	A file descriptor to assign (or the path of a file to open) as the default output file

Details

The path of the file to open may be absolute or relative to the current working directory.

When accessed from a remote node using the TSP-Link network, this command does not accept a file descriptor parameter and does not return a value.

If the function fails, an error message is returned.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** io.output")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
print(fileHandle)
end
io.close(fileHandle)
print(fileHandle)
os.remove(fileName)
```

Assign the file to be the default output file.

Also see

io.input() (on page 178)
io.open() (on page 180)

io.read()

This function reads data from the default input file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data1 = io.read()
data1 = io.read("format1")
data1, data2 = io.read("format1", "format2")
data1, ..., dataN = io.read("format1", ..., "formatN")
```

data1	The first data read from the file
data2	The second data read from the file
dataN	The last data read from the file; the number of return values matches the number of format values given
format1	A string or number indicating the type of the first data to be read
format2	A string or number indicating the type of the second data to be read
formatN	A string or number indicating the type of the final data to be read
...	One or more entries (or values) separated by commas

Details

The format parameters may be any of the following:

Format parameter	Description
"*n"	Returns a number
"*a"	Returns the whole file, starting at the present position; returns an empty string if it is at the end of file
"*l"	Default setting; returns the next line and skips the end of line character; returns nil if the present file position is at the end of file
N	Returns a string with the number of characters specified by *N*: ▪ If *N* is 5, 5 characters are returned ▪ If fewer than *N* characters are available, all characters are returned ▪ If *N* is zero (0), returns an empty string ▪ If the present file position is at the end of file, returns nil

Any number of format parameters may be passed to this command, each corresponding to a returned data value.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
-- io.read
print("\n*** io.read")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1\n")
myfile:flush()
myfile:close()
do
fileHandle = io.input(fileName)
value = io.read("*a")
print(value)
end
fileHandle:close()
print(eventlog.next())
```

Read data from the default input file.

Also see

None

io.type()

This function checks whether or not a given object is a file handle.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

type = io.type(*obj*)

type	Indicates whether the object is an open file handle
obj	Object to check

Details

Returns the string "file" if the object is an open file handle. If it is not an open file handle, nil is returned.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** io.type")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
state = io.type(fileHandle)
print(state)
end
io.close(fileHandle)
local state = io.type(fileHandle)
print(state)
os.remove(fileName)
```

Check whether or not `fileName` is a file handle.

Also see

io.open() (on page 180)

io.write()

This function writes data to the default output file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
io.write()
io.write(data1)
io.write(data1, data2)
io.write(data1, ..., dataN)
```

data1	The data to be written
data2	The data to be written
dataN	The data to be written
...	One or more values separated by commas

Details

All data parameters must be either strings or numbers.

NOTE: Data is not immediately written to a file when you use the `io.write()` function. The `io.write()` function buffers data; it may not be written to the USB flash drive immediately. Use the `io.flush()` function to immediately write buffered data to the drive.

Example

```
local fileName = "/usb1/myfile.txt"
if fs.is_file(fileName) then
        os.remove(fileName)
        print("Removing file")
else
        print("Nothing removed")
end
eventlog.clear()
print("\n*** io.write")
myfile, myfile_err, myfile_errnum = io.open(fileName, "w")
myfile:write("Line 1")
myfile:close()
do
fileHandle = io.output(fileName)
io.write("Line 2")
end
io.close(fileHandle)
os.remove(fileName)
```

Writes data to the default output file.

Also see

io.flush() (on page 177)

lan.applysettings()

This function re-initializes the LAN interface with new configuration settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.applysettings()
```

Details

Disconnects all existing LAN connections to the instrument and re-initializes the LAN with the present configuration settings. All existing socket sessions, such as HiSLIP or raw socket, will also be disconnected. If you are using TSP Toolkit, you must start a new session.

This function initiates a background operation. LAN configuration can be a lengthy operation. Although the function returns immediately, the LAN initialization continues to run in the background.

Even though the LAN configuration settings may not have changed since the LAN was last connected, new settings may take effect due to the dynamic nature of dynamic host configuration protocol (DHCP) or dynamic link local addressing (DLLA) configuration.

Re-initialization takes effect even if the configuration has not changed since the last time the instrument connected to the LAN.

NOTE: Wait approximately 10 s to 20 s after sending this command for the new settings to be active.

Example

```
lan.config.method = lan.MANUAL
lan.config.ipaddress = "192.0.2.18"
lan.config.gateway = "192.0.2.19"
lan.config.subnetmask = "255.255.255.0"
lan.applysettings()
```

Set up LAN using manually specified settings and re-initialize the LAN interface with new settings.

Also see

None

lan.autoconnect

This attribute is used to enable or disable link monitoring.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset lan.restoredefaults()	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.autoconnect
lan.autoconnect = state
```

state	LAN link monitoring state: ▪ Enables automatic link reconnection and monitoring: `localnode.ENABLE` ▪ Disables automatic link reconnection and monitoring: `localnode.DISABLE`

Details

This attribute sets the LAN link monitoring and automatic connection state.

When autoconnect is enabled, all connections are closed if the link to the LAN is lost for more than the time specified by `lan.linktimeout`.

Enable this attribute to automatically reset the LAN connection after the LAN link is established.

Example

```
lan.autoconnect = localnode.ENABLE
lan.applysettings()
```

Enable LAN link monitoring.

Also see

lan.linktimeout (on page 195)
lan.restoredefaults() (on page 198)

lan.config.dns.address[N]

This command configures the DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset lan.restoredefaults()	Nonvolatile memory	"0.0.0.0"

Usage

```
dnsAddress = lan.config.dns.address[N]
lan.config.dns.address[N] = "dnsAddress"
```

dnsAddress	DNS server IP address
N	Entry index: 1 or 2

Details

This attribute is an array of Domain Name System (DNS) server addresses. These addresses take priority for DNS lookups and are consulted before any server addresses that are obtained using DHCP. This allows local DNS servers to be specified that take priority over DHCP-configured global DNS servers.

You can specify up to two addresses. The address specified by `1` is consulted first for DNS lookups. *dnsAddress* must be a string specifying the IP address of the DNS server in dotted decimal notation.

Unused entries are returned as `0.0.0.0` when read. To disable an entry, set its value to `0.0.0.0` or the empty string `""`.

Although only two addresses may be manually specified here, the instrument uses up to three DNS server addresses. If two are specified here, only one that is given by a DHCP server is used. If no entries are specified here, up to three addresses that are given by a DHCP server are used.

Example

```
dnsaddress = "192.0.2.18"
lan.config.dns.address[1] = dnsaddress
lan.applysettings()
```

Set the DNS address 1 to `192.0.2.18`.

Also see

lan.config.dns.domain (on page 187)
lan.config.dns.dynamic (on page 188)
lan.config.dns.hostname (on page 189)
lan.config.dns.verify (on page 189)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

lan.config.dns.domain

This attribute configures the dynamic DNS domain.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset lan.restoredefaults()	Nonvolatile memory	""

Usage

```
domain = lan.config.dns.domain
lan.config.dns.domain = "domain"
```

domain	Dynamic DNS registration domain; a string of 255 characters or less

Details

This attribute holds the domain to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the domain specified in this attribute with the DNS server.

The length of the fully qualified host name (combined length of the domain and host name with separator characters) must be less than or equal to 255 characters. Although up to 255 characters are allowed, you must make sure the combined length is also no more than 255 characters.

Example

```
print(lan.config.dns.domain)
```

Outputs the present dynamic DNS domain. For example, if the domain is `Matrix`, the response is:

```
Matrix
```

Also see

lan.config.dns.dynamic (on page 188)
lan.config.dns.hostname (on page 189)
lan.config.dns.verify (on page 189)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

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.reset lan.restoredefaults()	Nonvolatile memory	localnode.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: ▪ Enabled: `localnode.ENABLE` ▪ Disabled: `localnode.DISABLE`

Details

Dynamic DNS registration works with DHCP to register the host name with the DNS server. The host name is specified in the `lan.config.dns.hostname` attribute.

Example

```
print(lan.config.dns.dynamic)
```

Outputs the dynamic registration state.

If dynamic DNS registration is enabled, the response is:

```
ENABLE
```

Also see

lan.config.dns.hostname (on page 189)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

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	Front-panel LAN settings	Nonvolatile memory	Instrument specific (see Details)

Usage

```
hostName = lan.config.dns.hostname
lan.config.dns.hostname = "hostName"
```

hostName	The host name to use for dynamic DNS registration; the host name must be a string of ▪ 63 characters or less ▪ start with a letter ▪ end with a letter or digit ▪ contain only letters, digits, and hyphens

Details

This attribute holds the host name to request during dynamic DNS registration. Dynamic DNS registration works with DHCP to register the host name specified in this attribute with the DNS server.

The factory default value for *hostName* is "k-*xxxxxx*-*#######*"), where *xxxxxx* is the model number and *########* is the mainframe serial number. Note that a hyphen separates 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 63 characters can be entered here, you must make sure the combined length is no more than 255 characters.

Setting this attribute to an empty string (in other words, setting this attribute to a string of length zero or a string that consists entirely of whitespace characters) reverts the host name to the factory default value.

Example

```
lan.config.dns.hostname = "Tester200"
lan.applysettings()
print(lan.config.dns.hostname)
```

Outputs the present dynamic DNS host name. For example:

```
Tester200
```

Also see

lan.config.dns.domain (on page 187)
lan.config.dns.dynamic (on page 188)
lan.restoredefaults() (on page 198)

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.reset lan.restoredefaults()	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.config.dns.verify
lan.config.dns.verify = state
```

state	DNS hostname verification state: ▪ DNS host name verification enabled: `localnode.ENABLE` ▪ DNS host name verification disabled: `localnode.DISABLE`

Details

When this is enabled, the instrument performs DNS lookups to verify that the DNS host name matches the value specified by `lan.config.dns.hostname`.

Example

```
print(lan.config.dns.verify)
```

Outputs the present DNS host name verification state.

If it is enabled, the output is:

```
ENABLE
```

Also see

lan.config.dns.hostname (on page 189)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

lan.config.gateway

This attribute contains the LAN default gateway address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset lan.restoredefaults()	Nonvolatile memory	"0.0.0.0"

Usage

```
gatewayAddress = lan.config.gateway
lan.config.gateway = "gatewayAddress"
```

gatewayAddress	LAN default gateway address; must be a string specifying the default IP address of the gateway in dotted decimal notation

Details

This attribute specifies the default gateway IP address to use when manual or dynamic link local addressing (DLLA) configuration methods are used to configure the LAN. If DHCP is enabled, this setting is ignored.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.gateway` attribute to determine the present operating state of the LAN.

The IP address must be formatted in four groups of numbers, each separated by a decimal.

Example

```
lan.config.method = lan.MANUAL
lan.config.ipaddress = "192.0.2.18"
lan.config.gateway = "192.0.2.19"
lan.config.subnetmask = "255.255.255.0"
lan.applysettings()
```

Set up LAN using manually specified settings and re-initialize the LAN interface with new settings.

Also see

lan.reset() (on page 198)

lan.restoredefaults() (on page 198)
lan.status.gateway (on page 201)

lan.config.ipaddress

This command specifies the LAN IP address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset() lan.restoredefaults()	Nonvolatile memory	"0.0.0.0" or "192.168.0.2"

Usage

```
ipAddress = lan.config.ipaddress
lan.config.ipaddress = "ipAddress"
```

ipAddress	LAN IP address; must be a string specifying the IP address in dotted decimal notation

Details

This command specifies the LAN IP address to use when the LAN is configured using the manual configuration method. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting that is presently in effect. Use the `lan.status.ipaddress` attribute to determine the present operating state of the LAN.

Example

```
lan.config.method = lan.MANUAL
lan.config.ipaddress = "192.0.2.18"
lan.config.gateway = "192.0.2.19"
lan.config.subnetmask = "255.255.255.0"
lan.applysettings()
```

Set up LAN using manually specified settings and re-initialize the LAN interface with new settings.

Also see

lan.config.method (on page 191)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)
lan.status.ipaddress (on page 201)

lan.config.method

This attribute contains the LAN settings configuration method.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset() lan.restoredefaults()	Nonvolatile memory	lan.AUTO

Usage

```
method = lan.config.method
lan.config.method = method
```

method	The method for configuring LAN settings; it can be one of the following values: ▪ Selects automatic sequencing of configuration methods: `lan.AUTO` ▪ Use only manually specified configuration settings: `lan.MANUAL`

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

```
lan.config.method = lan.MANUAL
lan.config.ipaddress = "192.0.2.18"
lan.config.gateway = "192.0.2.19"
lan.config.subnetmask = "255.255.255.0"
lan.applysettings()
```

Set up LAN using manually specified settings and re-initialize the LAN interface with new settings.

Also see

lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

lan.config.subnetmask

This attribute contains the LAN subnet mask.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset() lan.restoredefaults()	Nonvolatile memory	"255.255.255.0"

Usage

```
mask = lan.config.subnetmask
lan.config.subnetmask = "mask"
```

mask	String that specifies the LAN subnet mask value in dotted decimal notation

Details

This attribute specifies the LAN subnet mask that is used when the manual configuration method is used to configure the LAN. This setting is ignored when DLLA or DHCP is used.

This attribute does not indicate the actual setting presently in effect. Use the `lan.status.subnetmask` attribute to determine the present operating state of the LAN.

Example

```
lan.config.method = lan.MANUAL
lan.config.ipaddress = "192.0.2.18"
lan.config.gateway = "192.0.2.19"
lan.config.subnetmask = "255.255.255.0"
lan.applysettings()
```

Set up LAN using manually specified settings and re-initialize the LAN interface with new settings.

Also see

lan.reset() (on page 198)
lan.restoredefaults() (on page 198)
lan.status.subnetmask (on page 205)

lan.enable

This attribute controls whether or not any communications using the LAN connector are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN reset	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.enable
lan.enable = state
```

state	LAN connector communications state: ▪ Enable communications through the LAN connector: `localnode.ENABLE` ▪ Disable communications through the LAN connector: `localnode.DISABLE`

Details

This is the master LAN control setting. When this is enabled, you may individually control DST, HiSLIP, raw socket, and telnet access to the instrument. However, when this is disabled, all LAN communications are disabled, which overrides the individual LAN-enabled settings.

To disable only certain LAN communications with the instrument, enable this attribute and set the specific LAN communications attribute to false for DST, HiSLIP, raw sockets, or telnet.

If LAN communications are disabled while using LAN, you must use USBTMC or the front panel to enable LAN again.

When this attribute is set to `localnode.DISABLE`, `lan.identify` is automatically set to `localnode.OFF`.

Example 1

```
lan.enable = localnode.ENABLE
```

Enables LAN communications through the LAN connector on the mainframe.

Also see

lan.hislip.access (on page 193)
lan.identify (on page 194)
lan.status.port.dst (on page 202)
lan.status.port.rawsocket (on page 203)
lan.status.port.telnet (on page 204)

lan.hislip.access

This command enables or disables mainframe HiSLIP port communications.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults() lan.enable (see Details)	Nonvolatile memory	localnode.PROTECT

Usage

```
state = lan.hislip.access
lan.hislip.access = state
```

state	HiSLIP port communications state: ▪ Enable password-protected HiSLIP communications: `localnode.PROTECT` ▪ Enable HiSLIP communications: `localnode.ENABLE` ▪ Disable HiSLIP communications: `localnode.DISABLE`

Details

Password-enabled communications are enabled by default and authorized by the `login` command. In this state, you must log in to the mainframe to enable HiSLIP communications if the instrument is restarted or the `logout` command is sent.

When set to `localnode.ENABLE`, this command enables communications through the HiSLIP port (no password needed). When this command is set to `localnode.DISABLE`, all HiSLIP communications are disabled.

This command only takes effect when `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.hislip.access = localnode.ENABLE
```

Enables HiSLIP port communications.

Also see

lan.enable (on page 193)
lan.rawsocket.access (on page 197)
lan.telnet.access (on page 205)
login (on page 213)
logout (on page 214)
usbtmc.access (on page 343)

lan.identify

This attribute controls the LXI LAN indicator on the front panel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	localnode.OFF

Usage

```
identify = lan.identify
lan.identify = identify
```

identify	LXI identify setting: ▪ The LAN indicator is flashing: `localnode.ON` ▪ The LAN indicator is steady on: `localnode.OFF`

Details

If you have a bank of mainframe instruments, you can use this command to determine which one you are communicating with.

When this attribute is set to `localnode.ON`, the LAN indicator on the front panel flashes to indicate which instrument is in the identify mode. When the attribute is set to `localnode.OFF`, the LAN indicator does not flash.

When `lan.enable` is set to `localnode.DISABLE`, this attribute is automatically set to `localnode.OFF`.

Example

```
lan.identify = localnode.ON
```

The LAN indicator on the front panel of the mainframe flashes.

Also see

lan.enable (on page 193)

lan.linktimeout

This attribute contains the LAN link timeout period.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults()	Nonvolatile memory	20 (20 s)

Usage

```
timeout = lan.linktimeout
lan.linktimeout = timeout
```

timeout	The LAN link monitor timeout period (in seconds)

Details

You must enable the command `lan.autoconnect` before you can use this attribute.

The *timeout* value represents the amount of time that passes before the instrument disconnects from the LAN due to the loss of the LAN link integrity.

The LAN interface does not disconnect if the connection to the LAN is reestablished before the *timeout* value expires.

If the LAN link integrity is not restored before the *timeout* value expires, the instrument begins to monitor for a new connection.

Example

```
lan.autoconnect = localnode.ENABLE
lan.linktimeout = 30
lan.applysettings()
```

Enable LAN autoconnect and set the link timeout time to 30 s.

Also see

lan.autoconnect (on page 186)
lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

lan.mdns.enable

This attribute controls whether or not multicast DNS (mDNS) communications are enabled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset() lan.restoredefaults()	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.mdns.enable
lan.mdns.enable = state
```

state	Multicast DNS communications state: ▪ Enable mDNS: `localnode.ENABLE` ▪ Disable mDNS : `localnode.DISABLE`

Details

When set to `localnode.ENABLE`, this command enables mDNS communications. When this command is set to `localnode.DISABLE`, all mDNS communications are disabled.

This command is only available if `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.mdns.enable = localnode.ENABLE
lan.applysettings()
```

Enables mDNS.

Also see

lan.enable (on page 193)
lan.hislip.access (on page 193)
lan.status.port.dst (on page 202)
lan.status.port.rawsocket (on page 203)
lan.status.port.telnet (on page 204)

lan.nagle

This attribute controls the state of the LAN Nagle algorithm.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults()	Nonvolatile memory	localnode.DISABLE

Usage

```
state = lan.nagle
lan.nagle = state
```

state	The state of the Nagle algorithm: ▪ Enable the LAN Nagle algorithm for TCP connections: `localnode.ENABLE` ▪ Disable the Nagle algorithm for TCP connections: `localnode.DISABLE`

Details

The LAN Nagle algorithm increases efficiency by combining small packets before transmission. This attribute enables or disables the use of the LAN Nagle algorithm on transmission control protocol (TCP) connections.

Example

```
print(lan.nagle)
```

Outputs the present state of the LAN Nagle algorithm.

Also see

lan.reset() (on page 198)
lan.restoredefaults() (on page 198)

lan.ping.enable

This attribute determines if the instrument responds to a network ping request.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings lan.reset() lan.restoredefaults()	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.ping.enable
lan.ping.enable = state
```

state	Network ping response state: ▪ Enable ping responses: `localnode.ENABLE` ▪ Disable ping responses: `localnode.DISABLE`

Details

When this attribute is set to `localnode.ENABLE`, the instrument responds to network ping requests. When this command is set to `localnode.DISABLE`, the instrument does not respond to ping requests.

This command is only available if `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.ping.enable = localnode.ENABLE
lan.applysettings()
```

Configures the instrument to respond to a ping request.

Also see

lan.enable (on page 193)

lan.rawsocket.access

This command enables or disables mainframe raw socket port communications

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults() lan.enable (see Details)	Nonvolatile memory	localnode.PROTECT

Usage

```
state = lan.rawsocket.access
lan.rawsocket.access = state
```

state	Raw socket port communications state: ▪ Enable password-protected raw socket communications: `localnode.PROTECT` ▪ Enable raw socket communications: `localnode.ENABLE` ▪ Disable raw socket communications: `localnode.DISABLE`

Details

Password-enabled communications are enabled by default and authorized by the `login` command. In this state, you must log in to the mainframe to enable raw socket communications if the instrument is restarted or the `logout` command is sent.

When set to `localnode.ENABLE`, this command enables communications through the raw socket port (no password needed). When this command is set to `localnode.DISABLE`, all raw socket communications are disabled.

This command only takes effect when `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.enable = localnode.ENABLE
lan.rawsocket.access = localnode.ENABLE
```

Enables raw socket port communications.

Also see

lan.enable (on page 193)
lan.hislip.access (on page 193)
lan.telnet.access (on page 205)
login (on page 213)
logout (on page 214)
usb.tmc.access (on page 343)

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 runs the command `lan.restoredefaults()` and `lan.applysettings()`. This function also resets the LAN password and disables LAN communications.

NOTE: This function resets the password, which requires the user to specify a new password. This must be set from the front panel or by sending `user.password.change()` over a connection to the instrument. After setting the password, you must enable LAN communications with `lan.enable`.

To configure which LAN protocols use the password with the commands, use `lan.hislip.access`, `lan.rawsocket.access`, and `lan.telnet.access`.

Also see

lan.applysettings() (on page 185)
lan.restoredefaults() (on page 198)
user.password.change() (on page 343)

lan.restoredefaults()

This function resets the LAN commands to default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
lan.restoredefaults()
```

Details

The settings that are restored are shown in the following table.

Settings that are restored to default	
Attribute	**Default setting**
lan.autoconnect	localnode.ENABLE
lan.config.dns.address[N]	"0.0.0.0"
lan.config.dns.domain	""
lan.config.dns.dynamic	localnode.ENABLE
lan.config.dns.verify	localnode.ENABLE
lan.config.gateway	"0.0.0.0"
lan.config.ipaddress	"0.0.0.0" or "192.168.0.2"
lan.config.method	lan.AUTO
lan.config.subnetmask	"255.255.255.0"
lan.hislip.access	localnode.PROTECT
lan.linktimeout	20 (seconds)
lan.mdns.enable	localnode.ENABLE
lan.nagle	localnode.DISABLE
lan.ping.enable	localnode.ENABLE
lan.rawsocket.access	localnode.PROTECT
lan.telnet.access	localnode.PROTECT
usbtmc.access	localnode.ENABLE
lan.web.enable	localnode.ENABLE

This command is run when `lan.reset()` is sent.

If you are using TSP Toolkit, set `lan.rawsocket.access` and `lan.telnet.access` to `localnode.ENABLE`.

To apply the default settings, run `lan.applysettings()` after `lan.restoredefaults()`.

NOTE: This function resets the access to require a password for communications over LAN for `lan.hislip.access`, `lan.rawsocket.access`, and `lan.telnet.access`. The `lan.restoredefaults()` function does not reset the LAN password. The `user.password.change()` attribute controls the LAN password, which can be reset with `lan.reset()`.

Example

```
lan.restoredefaults()
lan.applysettings()
```

Restores the LAN defaults.

Also see

lan.reset() (on page 198)

lan.status.dns.address[N]

This attribute contains the DNS server IP addresses.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
dnsAddress = lan.status.dns.address[N]
```

dnsAddress	DNS server IP address
N	Entry index: 1, 2, or 3

Details

This attribute is an array of Domain Name System (DNS) server addresses. The instrument can use up to three addresses.

Unused or disabled entries are returned as `"0.0.0.0"` when read. The *dnsAddress* returned is a string specifying the IP address of the DNS server in dotted decimal notation.

The value of `lan.status.dns.address[1]` is referenced first for all DNS lookups. The values of `lan.status.dns.address[2]` and `lan.status.dns.address[3]` are referenced second and third, respectively.

Example

```
print(lan.status.dns.address[1])
```

Outputs the DNS server address 1, for example:

```
192.0.2.18
```

Also see

lan.config.dns.address[N] (on page 187)
lan.status.dns.name (on page 200)

lan.status.dns.name

This attribute contains the present DNS fully qualified host name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
hostName = lan.status.dns.name
```

hostName	Fully qualified DNS host name that can be used to connect to the instrument

Details

A fully qualified domain name (FQDN) specifies its exact location in the tree hierarchy of the Domain Name System (DNS).

A FQDN is the complete domain name for a specific computer or host on the LAN. The FQDN consists of two parts: The host name and the domain name.

If the DNS host name for an instrument is not found, this attribute stores the IP address in dotted decimal notation.

Example

```
print(lan.status.dns.name)
```

Outputs the dynamic DNS host name.

Also see

lan.config.dns.domain (on page 187)
lan.config.dns.hostname (on page 189)

lan.status.gateway

This attribute contains the gateway address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

gatewayAddress = lan.status.gateway

gatewayAddress	LAN gateway address presently in use

Details

The value of *gatewayAddress* is a string that indicates the IP address of the gateway in dotted decimal notation.

Example

```
print(lan.status.gateway)
```

Outputs the gateway address, such as:

```
192.168.0.1
```

Also see

lan.config.gateway (on page 190)

lan.status.ipaddress

This attribute contains the LAN IP address presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

ipAddress = lan.status.ipaddress

ipAddress	LAN IP address specified in dotted decimal notation

Details

The IP address is a character string that represents the IP address assigned to the instrument.

Example

```
print(lan.status.ipaddress)
```

Outputs the LAN IP address currently in use, such as:

```
192.168.0.2
```

Also see

lan.config.ipaddress (on page 191)

lan.status.macaddress

This attribute contains the LAN MAC address.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
macAddress = lan.status.macaddress
```

macAddress	The instrument MAC address

Details

The MAC address is a character string representing the MAC address of the instrument in hexadecimal notation. The string includes colons that separate the address octets (see **Example**).

Example

```
print(lan.status.macaddress)
```

Outputs the MAC address of the instrument, for example:

```
08:00:11:00:00:57
```

Also see

None

lan.status.port.dst

This attribute contains the LAN dead socket termination (DST) port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.dst
```

port	DST port number: Always 5030

Details

This attribute holds the TCP port number used to reset all other LAN socket connections.

To reset all LAN connections, open a connection to the DST port number.

Example

```
print(lan.status.port.dst)
```

Outputs the LAN DST port number, such as:

```
5.03000e+03
```

Also see

None

lan.status.port.hislip

This attribute contains the HiSLIP 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.hislip
```

port	HiSLIP port number: Always 4880

Details

This attribute stores the TCP port number used to connect to the instrument over a HiSLIP interface.

Example

```
print(lan.status.port.hislip)
```

Outputs the HiSLIP number, such as:

```
4.88000e+03
```

Also see

None

lan.status.port.rawsocket

This attribute contains the LAN raw socket connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.rawsocket
```

port	Raw socket port number: Always 5025

Details

The TCP port number is used to connect and control the instrument over a raw socket communications interface.

Example

```
print(lan.status.port.rawsocket)
```

Outputs the LAN raw socket port number, such as:

```
5.02500e+03
```

Also see

None

lan.status.port.telnet

This attribute contains the LAN telnet connection port number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
port = lan.status.port.telnet
```

port	Telnet port number: Always 23

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)
```

Outputs the LAN telnet connection port number.

Output:

```
2.3000000e+01
```

Also see

None

lan.status.speed

This attribute contains the LAN speed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
speed = lan.status.speed
```

speed	LAN speed in Mbps: 10, 100, or 1000

Details

This attribute indicates the transmission speed presently in use by the LAN interface.

Example

```
print(lan.status.speed)
```

Outputs the transmission speed of the instrument presently in use, such as:

```
1.0000000e+02
```

Also see

None

lan.status.subnetmask

This attribute contains the LAN subnet mask that is presently in use by the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
mask = lan.status.subnetmask
```

mask	A string specifying the subnet mask in dotted decimal notation

Details

Use this attribute to determine the present operating state of the LAN. This attribute returns the present LAN subnet mask value if the LAN is manually configured or when DLLA or DHCP is used.

Example

```
print(lan.status.subnetmask)
```

Outputs the subnet mask of the instrument that is presently in use, such as:

```
255.255.255.0
```

Also see

lan.config.subnetmask (on page 192)

lan.telnet.access

This command is used to enable or disable mainframe telnet port communications

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults lan.enable (see Details)	Nonvolatile memory	localnode.PROTECT

Usage

```
state = lan.telnet.access
lan.telnet.access = state
```

state	Telnet port communications state: ▪ Enable password-protected telnet communications: `localnode.PROTECT` ▪ Enable telnet communications: `localnode.ENABLE` ▪ Disable telnet communications: `localnode.DISABLE`

Details

Password-enabled communications are enabled by default and authorized by the `login` command. In this state, you must log in to the mainframe to enable telnet communications if the instrument is restarted or the `logout` command is sent.

When set to `localnode.ENABLE`, this command enables communications through the telnet port (no password needed). When this command is set to `localnode.DISABLE`, all telnet communications are disabled.

This command only takes effect if `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.enable = localnode.ENABLE
lan.telnet.access = localnode.ENABLE
```

Enables telnet port communications.

Also see

lan.enable (on page 193)
lan.hislip.access (on page 193)
lan.rawsocket.access (on page 197)
login (on page 213)
logout (on page 214)
usbtmc.access (on page 343)

lan.web.enable

This command enables or disables the mainframe web interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults lan.enable (see Details)	Nonvolatile memory	localnode.ENABLE

Usage

```
state = lan.web.enable
lan.web.enable = state
```

state	Mainframe web page interface state: ▪ Enable web page interface: `localnode.ENABLE` ▪ Disable web page interface: `localnode.DISABLE`

Details

When this attribute is set to `localnode.DISABLE`, the web interface is disabled and is not accessible. The following services and applications are also affected:

- The LXI Identification `.xml` page is not accessible.
- TSP Toolkit cannot connect over raw socket or HiSLIP because TSP Toolkit requires the LXI Identification `.xml` page.
- NI-VISA will not find the instrument through automatic discovery. Other VISA clients may find the instrument through automatic discovery.

This command only takes effect when `lan.enable` is set to `localnode.ENABLE`.

Example

```
lan.web.enable = localnode.DISABLE
```

Disables web page access for the mainframe.

Also see

lan.enable (on page 193)
lan.rawsocket.access (on page 197)
lan.telnet.access (on page 205)
login (on page 213)
logout (on page 214)

localnode.autolinefreq

This attribute enables or disables automatic power line frequency detection at startup.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.ENABLE

Usage

```
state = localnode.autolinefreq
localnode.autolinefreq = state
```

flag	The automatic line frequency detection state: • `localnode.ENABLE`: Line frequency is automatically detected at startup • `localnode.DISABLE`: Line frequency detection is disabled at startup

Details

When this attribute is set to `ENABLE`, the power line frequency of 50 Hz or 60 Hz is detected automatically the next time the mainframe powers up. If the power line frequency is automatically detected, the `localnode.linefreq` attribute is set to 50 or 60 (Hz). If the frequency cannot be detected or if an error occurs, it defaults to 60 Hz.

If the `localnode.linefreq` attribute is set, this attribute is automatically set to `localnode.DISABLE`.

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].autolinefreq = localnode.ENABLE`.

Example

```
localnode.autolinefreq = localnode.DISABLE
```

Disables automatic line frequency detection for the next mainframe power-up.

Also see

localnode.linefreq (on page 208)

localnode.description

This attribute stores a user-defined description and mDNS service name of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Front-panel LAN settings	Nonvolatile memory	See Details

Usage

```
localnode.description = "description"
description = localnode.description
```

description	User-defined description and mDNS service name of the instrument; use a string of 63 characters or less

Details

This attribute stores a string that contains a description of the instrument. The value of this attribute is also used as the mDNS service name of the instrument.

The factory default value of this attribute is "Tektronix Modular Mainframe MP5103 – *xxxxxxx*", where *xxxxxxx* is replaced with the serial number of the instrument. To revert this description to the factory default values, set this attribute to an empty string (a string of length zero).

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].description`.

Example

```
localnode.description = "My MP5103"
print(localnode.description)
```

Outputs the user-defined description and mDNS host name. For example:

```
My MP5103
```

Also see

None

localnode.license

This attribute stores the licenses for the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

```
license = localnode.license
```

license	Licenses for the instrument

Details

This attribute contains a large amount of text, including the Tektronix End User License Agreement and any open source software licenses in effect..

Example

```
print(localnode.license)
```

Returns the licenses for the instrument.

Also see

localnode.model (on page 210)
localnode.revision (on page 212)

localnode.linefreq

This attribute contains the power line frequency that is either detected at power-on or manually set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	See Details	Nonvolatile memory	Automatically sensed, or 60 (60 Hz)

Usage

```
frequency = localnode.linefreq
localnode.linefreq = frequency
```

frequency	An integer (50 or 60) representing the detected or specified power line frequency of the mainframe in Hertz

Details

To achieve optimum noise rejection when performing measurements at integer NPLC apertures, set the line frequency attribute to match the frequency (50 Hz or 60 Hz) of the ac power line.

When this attribute is set, the `localnode.autolinefreq` attribute is automatically set to `localnode.DISABLE`, and the returned value of this attribute is the value (50 Hz or 60 Hz) set by the user. User-specified values are preserved through a power-cycle.

If the `localnode.autolinefreq` attribute is set to `localnode.ENABLE`, this attribute will return the frequency detected by the mainframe. Power-cycling the unit will auto-detect the frequency again. If the line frequency cannot be detected or if an error occurs, it defaults to 60 Hz.

When using this command from a remote node, replace `localnode` with the node reference, for example `node[5].linefreq = 60`.

Example 1

```
frequency = localnode.linefreq
print(frequency)
```

Reads the power line frequency

Example 2

```
localnode.linefreq = 50
```

Sets the power line frequency to 50 Hz.

Also see

localnode.autolinefreq (on page 207)

localnode.manufacturer

This attribute stores the manufacturer of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

manufacturer = localnode.manufacturer

manufacturer	Always `TEKTRONIX`

Example

```
print(localnode.manufacturer)
```

Outputs the manufacturer:

```
TEKTRONIX
```

Also see

None

localnode.model

This attribute stores the model number of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

```
model = localnode.model
```

model	The model number of the mainframe

Details

This attribute stores the model number of the mainframe.

NOTE: To retrieve the model number of a module, use `slot[Z].model`.

Example

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

Outputs the model number of the local node.

Example output:

```
MP5103
```

Also see

localnode.serialno (on page 212)

localnode.prompts

This attribute determines if the instrument generates prompts in response to command messages.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	localnode.ENABLE

Usage

```
prompting = localnode.prompts
localnode.prompts = prompting
```

prompting	Prompting mode ▪ Generate prompts: `localnode.ENABLE` ▪ Do not generate prompts: `localnode.DISABLE`

Details

When the prompting mode is enabled, the instrument generates prompts when the instrument is ready to take another command. Because the prompt is not generated until the previous command completes, enabling prompts provides handshaking with the instrument to prevent buffer overruns.

When prompting is enabled, the instrument might generate the following prompts:

- **TSP>**. The standard prompt, which indicates that the previous command completed normally.
- **TSP?**. The prompt that is issued if there are unread entries in the event log when the prompt is issued. Like the TSP> prompt, it indicates that processing of the command is complete. It does not mean the previous command generated an error, only that there were still errors in the queue when the command processing was complete.
- **>>>>**. The continuation prompt, which occurs when downloading scripts. When downloading scripts, many command messages must be sent as a group. The continuation prompt indicates that the instrument is expecting more messages as part of the present command.

Commands do not generate prompts. The instrument generates prompts in response to command completion.

Prompts are enabled or disabled only for the remote interface that is active when you send the command. For example, if you enable prompts when the LAN connection is active, they are not enabled for a subsequent USB connection.

Example

```
localnode.prompts = localnode.ENABLE
```

Generate prompts.

Also see

localnode.prompts4882 (on page 211)
localnode.showerrors (on page 213)

localnode.prompts4882

This attribute enables or disables the generation of prompts for IEEE Std 488.2 common commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	localnode.ENABLE

Usage

```
prompting = localnode.prompts4882
localnode.prompts4882 = prompting
```

prompting	Prompting mode ▪ Generate prompts: `localnode.ENABLE` ▪ Do not generate prompts: `localnode.DISABLE`

Details

When this attribute is enabled, the IEEE Std 488.2 common commands generate prompts if prompting is also enabled with the `localnode.prompts` attribute. If `localnode.prompts4882` is enabled, limit the number of `*TRG` commands sent to a running script to 50 regardless of the setting of the `localnode.prompts` attribute.

When this attribute is disabled, IEEE Std 488.2 common commands do not generate prompts. When using the `*TRG` command with a script that executes `trigger.wait()` repeatedly, disable prompting to avoid problems associated with the command interface input queue filling.

Example

```
localnode.prompts = localnode.ENABLE
localnode.prompts4882 = localnode.ENABLE
```

Enables IEEE Std 288.2 common command prompting.

Also see

localnode.prompts (on page 210)

localnode.revision

This attribute stores the firmware revision level of the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
revision = localnode.revision
```

revision	The firmware revision number of the instrument

Example

```
print(localnode.revision)
```

Outputs the firmware revision number of the instrument

Also see

localnode.manufacturer (on page 209)
localnode.model (on page 210)
localnode.serialno (on page 212)

localnode.serialno

This attribute stores the serial number of the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
serialno = localnode.serialno
```

serialno	The serial number of the instrument

Example

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

Returns the serial number of the instrument

Also see

localnode.model (on page 210)
localnode.revision (on page 212)

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)	No	Not applicable	Not applicable	localnode.DISABLE

Usage

```
errorMode = localnode.showerrors
localnode.showerrors = errorMode
```

errorMode	Show error setting: ▪ Send errors: `localnode.ENABLE` ▪ Do not send errors: `localnode.DISABLE`

Details

If this attribute is set to enable, the instrument automatically sends any generated errors stored in the event log, and then clears the log. Errors are processed after executing a command message (just before issuing a prompt if prompts are enabled).

If this attribute is set to disable, errors are left in the event log and must be explicitly read or cleared.

When using this command from a remote node, replace `localnode` with the node reference, for example, `node[5].showerrors`.

Example

```
localnode.showerrors = localnode.ENABLE
```

Enables generated errors to be sent.

Also see

localnode.prompts (on page 210)
localnote.prompts4882 (on page 211)

login

This command logs a user in to the mainframe and allows remote access.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
login password
```

password	A string that represents the login password; the default value is no password

Details

This command logs a user into the mainframe and allows remote access. A user remains logged in until the instrument is reset, the mainframe is rebooted, or the `logout` command is sent.

The password can be added or changed with the `user.password.change` command.

Logins do not persist throughout mainframe and module firmware updates.

Example

```
login pass123
logout
```

Logs a user into the mainframe using password `pass123` and then logs out.

Also see

logout (on page 214)
user.password.change (on page 343)

logout

This function logs a user out of the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
logout
```

Details

This command logs a user out of the mainframe on the communications interface being used. To restart communications, you must login again. There is no effect if the interface used is set to `localnode.ENABLE`.

Example

```
logout
```

Logs a user out of the mainframe.

Also see

login (on page 213)
user.password.change (on page 343)

makegetter()

This function creates a function to get the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
getter = makegetter(table, "attributeName")
```

getter	Function that returns the value of the defined attribute
table	Read-only table where the attribute is located
attributeName	A string representing the name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the function created with `makegetter()` executes more quickly 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

```
getlevel = makegetter(slot[1].smu[1].source, "levelv")
v = getlevel()
```

Creates a getter function called `getlevel`.

When `getlevel()` is called, it returns the value of `slot[1].smu[1].source.levelv`, or the voltage level of channel 1 of the SMU installed in slot 1.

Also see

makesetter() (on page 215)

makesetter()

This function creates a function that, when called, sets the value of an attribute.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

setter = makesetter(*table*, "*attributeName*")

setter	Function that sets the value of the attribute
table	Read-only table where the attribute is located
attributeName	The string name of the attribute

Details

This function is useful for aliasing attributes to improve execution speed. Calling the *setter* function executes more quickly than accessing the attribute directly.

Creating a *setter* function is only useful if it is going to be called several times. If you are not calling the *setter* function several times, it is more efficient to access the attribute directly.

Example

```
setlevel = makesetter(slot[1].smu[1].source, "levelv")
for v = 1, 10 do
   setlevel(v)
end
```

Creates a setter function called `setlevel`.

Using `setlevel()` in the loop sets the value of `slot[1].smu[1].source.levelv`, performing a source sweep on channel 1 of the SMU installed in slot 1.

Also see

makegetter() (on page 214)

node[N].execute()

This function starts test scripts from a remote TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes (see Details)			

Usage

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

N	The node number of the specified instrument
scriptCode	A string containing the source code

Details

Only the remote master node can use the execute command to run a script on this node. This function does not run test scripts on the master node; only on node *N* when initiated by the master node.

This function may only be called when the group number of the node is different than the node of the master.

This function does not wait for the script to finish execution.

This command should only be used from a remote master when controlling this instrument over a TSP-Link™ connection.

Example 1

```
node[2].execute(sourcecode)
```

Runs script code on node 2. The code is in a string variable called `sourcecode`.

Example 2

```
node[3].execute("x = 5")
```

Runs script code in string constant (`"x = 5"`) to set `x` equal to 5 on node 3.

Example 3

```
node[32].execute(TestDut.source)
```

Runs the test script stored in the script variable `TestDut` (previously stored on the master node) on node 32.

Also see

tsplink.group (on page 315)

node[N].getglobal()

This function returns the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
value = node[N].getglobal("name")
```

value	The value of the variable
N	The node number of the instrument holding the variable: 1 to 64
name	The global variable name, as a string

Details

This function retrieves the value of a global variable from the runtime environment of this node.

Do not use this command to retrieve the value of a global variable from the local node. Instead, access the global variable directly. This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

```
print(node[5].getglobal("test_val"))
```

Retrieves and outputs the value of the global variable named `test_val` from node 5.

Also see

node[N].setglobal() (on page 217)

node[N].setglobal()

This function sets the value of a global variable.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
node[N].setglobal("name", value)
```

N	The node number of this instrument holding the global variable: 1 to 64
name	The global variable name to set, as a string
value	The value to assign to the variable

Details

From a remote node, use this function to assign the given value to a global variable.

Do not use this command to create or set the value of a global variable from the local node (set the global variable directly instead). This command should only be used from a remote master when controlling this instrument over a TSP-Link network.

Example

```
node[3].setglobal("x", 5)
```

Sets the global variable `x` on node 3 to the value of 5.

Also see

node[N].getglobal() (on page 217)

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 Status Event Status Register to be set when all previously started local overlapped and pending commands are complete.

Note that each node independently sets its operation complete bits in its own status model. Any nodes that are not actively performing overlapped commands set their bits immediately. All remaining nodes set their own bits as they complete their own overlapped commands.

Example

```
opc()
waitcomplete()
print(tostring(status.standard.event))
```

Output:

```
1
```

Also see

Status model (on page 634)

os.clock()

This function returns the number of seconds the instrument has been powered on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
seconds = os.clock()
```

seconds	The number of seconds that have elapsed since the instrument was powered on

Details

This function returns the total elapsed time since the instrument was powered on.

Example

```
t = os.clock()
```

Returns the elapsed time since the instrument was powered on.

Also see

os.time() (on page 220)

os.remove()

This function deletes the file or directory with a given name.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
success, msg = os.remove("filename")
```

success	A success indicator (`true` or `nil`)
msg	A message value (`nil` or an error message)
filename	A string representing the name of the file or directory to delete

Details

This path may be absolute or relative to the current working directory.

Directories must be empty before using the `os.remove()` function to delete them.

If this function fails, it returns `nil` (for *success*) and an error message string (for *msg*).

Example

```
os.remove("testFile")
```

Delete the file named `testFile`.

Also see

os.rename() (on page 219)

os.rename()

This function renames an existing file or directory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
success, msg = os.rename("oldname", "newname")
```

success	A success indicator: `true` or `nil`
msg	A message value: `nil` or an error message
oldname	String representing the name of the file or directory to rename
newname	String representing the new name of the file or directory

Details

If this function fails, it returns `nil` (for *`success`*) and an error message string (for *`msg`*).

Example

```
os.rename("testFile", "exampleFile")
```

Changes the name of the existing file `testFile` to the name `exampleFile`.

Also see

os.remove() (on page 219)

---

os.time()

This function generates a time value in UTC time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
utcSeconds, utcFractional = os.time()
utcSeconds, utcFractional = os.time(timespec)
```

`utcSeconds`	Time value in UTC time, which is the number of seconds since midnight January 1, 1970, UTC
`utcFractional`	When using the first form, this value holds the fractional seconds; when using the second form, this value is always 0
`timespec`	The date and time (year, month, day, hour, minute, and second)

Details

When used without parameters, this function returns the present time as the number of seconds since midnight January 1, 1970. The *`utcSeconds`* value is the elapsed seconds and *`utcFractional`* is the fractional seconds. Both values have the units of seconds. *`utcSeconds`* is an integer and *`utcFractional`* is a value between `0` and `1` but always less than `1`.

When using the second form, *`timespec`* is a table using the fields listed in the following table.

`year`	The year (`1970` or later)
`month`	The month (`1` to `12`)
`day`	The day (`1` to `31`)
`hour`	The hour (`0` to `23`)
`min`	The minute (`0` to `59`)
`sec`	The second (`0` to `59`)

If the time (*`hour`*, *`min`*, *`sec`*) fields are not provided, noon on that day is used. The time specified is given in the local time zone.

The local time zone should be set before using the `os.time()` function. The value returned when using the *`timespec`* parameter is the number of seconds elapsed between midnight January 1, 1970, and the specified date.

Example

```
systemTime = os.time({year = 2025, month = 3, day = 31, hour = 14, min = 25})
settime(systemTime)
```

Sets the date and time to Mar 31, 2019, at 2:25 pm.

Also see

os.clock() (on page 218)
settime() (on page 233)
settimezone() (on page 234)

print()

This function generates a response message.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
print(value1)
print(value1, value2)
print(value1, ..., valueN)
```

value1	The first argument to output
value2	The second argument to output
valueN	The last argument to output
...	One or more values separated with commas

Details

TSP-enabled instruments do not have inherent query commands. Like other scripting environments, the `print()` command and other related `print()` commands generate output. The `print()` command creates one response message.

The output from multiple arguments is separated with a tab character.

Numbers are printed using the `format.asciiprecision` attribute. If you want to use Lua formatting, print the return value from the `tostring()` function or set `format.asciiprecision` to zero (`0`).

Example 1

```
x = 10
print(x)
```

Example of an output response message:

```
1.00000e+01
```

Your output might be different, depending on the ASCII precision setting.

Example 2

```
x = true
print(tostring(x))
```

Example of an output response message:

```
true
```

Also see

format.asciiprecision (on page 169)
printbuffer() (on page 222)
printnumber() (on page 225)

printbuffer()

This function prints data from tables or reading buffer columns.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printbuffer(startIndex, endIndex, bufferVar)
printbuffer(startIndex, endIndex, bufferVar, bufferVar2)
printbuffer(startIndex, endIndex, bufferVar, ..., bufferVarN)
```

startIndex	Beginning index of the buffer to print; this must be more than one and less than *endIndex*
endIndex	Ending index of the buffer to print; this must be more than *startIndex* and less than the index of the last entry in the tables; you can use *bufferVar*.n
bufferVar	First table or reading buffer column to print
bufferVar2	Second table or reading buffer column to print
...	One or more tables or reading buffer columns separated with commas
bufferVarN	The last table or reading buffer column to print, where *N* is the table or reading buffer column

Details

This command generates a single response message that contains the requested data. The data is specified as a range of points and a set of columns of data.

The start and end indexes select a section of data points to return from the columns. Use *bufferVar*.n to specify the final reading in the buffer. For example, a start index of 1 and end index of *bufferVar*.n returns the entire column.

If overlapped commands use the specified reading buffer and data has not been generated to the specified end index, this function outputs data as it becomes available. When there are outstanding overlapped commands to acquire data, *endIndex* refers to the index of the last entry in the table after all the measurements have completed.

The data is returned in the order of the specified buffer columns. If only *bufferVar* is specified, the command returns readings only. Other buffer columns are available as shown in the following table.

The output of printbuffer() is affected by the data format selected by format.data. If you set format.data to format.REAL32 or format.REAL64, the only columns available to return are readings, statuses, and timestamps. If you request a column that is not allowed, the instrument returns 9.91e37.

Attribute	Description
bufferVar.basetimestampfractional	The fractional seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer. Refer to: ▪ PSU: bufferVar.basetimestampfractional (on page 349) ▪ SMU: bufferVar.basetimestampfractional (on page 444)
bufferVar.basetimestampseconds	The whole seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer. Refer to: ▪ PSU: bufferVar.basetimestampseconds (on page 349) ▪ SMU: bufferVar.basetimestampseconds (on page 445)
bufferVar.measurefunctions	The measurement functions that were used to acquire a reading stored in a specified reading buffer. Refer to: ▪ PSU: bufferVar.measurefunctions (on page 354) ▪ SMU: bufferVar.measurefunctions (on page 451)
bufferVar.measureranges	The measurement range values that were used for readings stored in a specified buffer. Refer to: ▪ PSU: bufferVar.measureranges (on page 355) ▪ SMU: bufferVar.measureranges (on page 452)
bufferVar.n	The number of readings in the buffer. Refer to: ▪ PSU: bufferVar.n (on page 355) ▪ SMU: bufferVar.n (on page 452)
bufferVar.readings	The readings stored in a specified reading buffer. Refer to: ▪ PSU: bufferVar.readings (on page 356) ▪ SMU: bufferVar.readings (on page 453)
bufferVar.sourcefunctions	The source functions that were used for readings stored in a specified reading buffer. Refer to: ▪ PSU: bufferVar.sourcefunctions (on page 357) ▪ SMU: bufferVar.sourcefunctions (on page 454)
bufferVar.sourceoutputstates	The states of the source output for readings stored in a specified buffer. Refer to: ▪ PSU: bufferVar.sourceoutputstates (on page 358) ▪ SMU: bufferVar.sourceoutputstates (on page 454)
bufferVar.sourceranges	The source range values for readings stored in the specified buffer. Refer to: ▪ PSU: bufferVar.sourceranges (on page 359) ▪ SMU: bufferVar.sourceranges (on page 457)
bufferVar.sourcevalues	The source levels that were being output when readings in the reading buffer were acquired. Refer to: ▪ PSU: bufferVar.sourcevalues (on page 360) ▪ SMU: bufferVar.sourcevalues (on page 458)
bufferVar.statuses	The status values of readings in the reading buffer. Refer to: ▪ PSU: bufferVar.statuses (on page 361) ▪ SMU: bufferVar.statuses (on page 459)

Attribute	Description
bufferVar.timestamps	The timestamps of readings stored in the reading buffer. Refer to: ▪ PSU: bufferVar.timestamps (on page 363) ▪ SMU: bufferVar.timestamps (on page 461)

Example

```
rb1 = slot[2].smu[1].makebuffer(10)
slot[2].smu[1].trigger.measure.v(rb1)
slot[2].smu[1].source.output = 0
slot[2].smu[1].source.levelv = 5
slot[2].trigger.model.delete("TM1")
slot[2].trigger.model.create("TM1")
-- Start the measurements.
slot[2].trigger.model.addblock.measure("TM1", "", 1, 10)
slot[2].trigger.model.addblock.delay.constant("TM1", "delay2", 0.05)
slot[2].trigger.model.addblock.source.output("TM1", "sourceoutput3", 1, 1)
slot[2].trigger.model.addblock.delay.constant("TM1", "delay4", 0.05)
slot[2].trigger.model.addblock.source.output("TM1", "sourceoutput5", 1, 0)
slot[2].trigger.model.initiate("TM1")
waitcomplete() print("Done")
printbuffer(1, rb1.n, rb1.readings, rb1.sourcefunctions, rb1.sourceoutputstates)
```

Create a reading buffer named rb1 for slot 2, SMU channel 1 measurements.

Define a voltage measurement that stores data in the rb1 reading buffer.

Set the source output off and set the source level to 0 V.

Create a trigger model named TM1.

Create a trigger model makes 10 measurements.

Return the readings, source functions, and source output states from `rb1`. The use of `rb1.n`

(*bufferVar*`.n`) indicates that the instrument should output all readings in the reading buffer. In this example, `rb1.n` equals 10.

Example of output data:

```
Done
-1.043131560e-04, Voltage, Off, -1.029029372e-04, Voltage, Off, -1.049783605e-04,
Voltage, Off, -1.073198582e-04, Voltage, Off, -1.037277834e-04, Voltage, Off, -1.0
19184419e-04, Voltage, Off, -1.060958894e-04, Voltage, Off, -1.062289302e-04, Volt
age, Off, -1.084107862e-04, Voltage, Off, -1.065482284e-04, Voltage, Off
```

Also see

format.asciiprecision (on page 169)
format.byteorder (on page 169)
format.data (on page 170)
print() (on page 221)
printnumber() (on page 225)

printnumber()

This function prints numbers using the configured format.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
printnumber(value1)
printnumber(value1, value2)
printnumber(value1, ..., valueN)
```

value1	First value to print in the configured format
value2	Second value to print in the configured format
valueN	Last value to print in the configured format
...	One or more values separated with commas

Details

There are multiple ways to use this function, depending on how many numbers are to be printed.

This function prints the given numbers using the data format specified by `format.data` and `format.asciiprecision`.

Example

```
format.asciiprecision = 10
x = 2.54
printnumber(x)
format.asciiprecision = 3
printnumber(x, 2.54321, 3.1)
```

Configure the ASCII precision to 10 and set `x` to 2.54.

Read the value of `x` based on these settings.

Change the ASCII precision to 3.

View how the change affects the output of `x` and some numbers.

Output:

```
2.540000000e+00
2.54e+00, 2.54e+00, 3.10e+00
```

Also see

format.asciiprecision (on page 169)
format.byteorder (on page 169)
format.data (on page 170)
print() (on page 221)
printbuffer() (on page 222)

reset()

This function resets commands to their default settings on the mainframe and all slots and channels.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reset()
reset(system)
```

system	What to reset in a TSP-Link system: ▪ To reset the entire system: `true` ▪ To reset only the local group: `false`

Details

If the instrument is on a TSP-Link network, `reset()` resets all nodes, including the controlling node and all subordinate nodes.

If the command is sent from the master, all instruments are reset.

If you send the command from a subordinate, `reset()` and `reset(true)` reset all instruments. `reset(false)` resets the local node group. If no groups exist, the entire network is considered to be a single group, and all nodes are reset.

If you want to reset a specific channel, use the slot-specific commands for your module, `slot[Z].psu[X].reset()` or `slot[Z].smu[X].reset()`.

Also see

slot[Z].psu[X].reset() (on page 391)
slot[Z].smu[X].reset() (on page 506)

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

Details

This command only applies to scripts that have been saved to the nonvolatile memory using `scriptVar.save()`. To remove a script that is in the runtime, set the variable containing the script to `nil` and run `collectgarbage()`.

Example

```
for name, value in pairs(script.table()) do
	script.delete(name)
end
```

Deletes all scripts from nonvolatile memory.

Also see

scriptVar.save() (on page 232)

script.load()

This function creates a script from a specified file.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.load("file")
scriptVar = script.load("file", "name")
```

scriptVar	The created script; this is `nil` if an error is encountered
file	The path and file name of the script file to load
name	The name of the script to be created

Details

The file path may be absolute or relative to the current working directory. The root folder of the USB flash drive has the absolute path `"/usb1/"`. Use the forward slash (/) for directory separators.

The file to be loaded must start with the `loadscript` keywords, contain the body of the script, and end with the `endscript` keyword.

Script naming:

- If the *name* parameter is an empty string, or *name* is absent (or `nil`) and the script name cannot be extracted from the file, *scriptVar* is the only handle to the created script.
- If *name* is given (and not `nil`), any script name embedded in the file is ignored.
- If *name* conflicts with the name of an existing script in `script.table()`, the name attribute of the existing script is set to an empty string before it is replaced in `script.table()` by the new script.
- If *name* is absent or `nil`, the command attempts to extract the name of the script from the file. Any conflict between the extracted name and that of an existing script in the scripts table generates an error. If the script name cannot be extracted, the name attribute of the created script is initialized to the empty string and must be set to a valid nonempty string before saving the script to nonvolatile memory.

Example

```
myTest8 = script.load("/usb1/filename.tsp", "myTest8")
```

Loads the script `myTest8` from the USB flash drive.

Also see

script.new() (on page 228)

script.new()

This function creates a script in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar = script.new("code")
scriptVar = script.new("code", "name")
```

scriptVar	The name of the variable that references the script
code	A string containing the body of the script
name	The name of the script

Details

The name parameter is the name that is added to `script.table()`. If a name is not provided, an empty string is used, and the script is unnamed. If the name already exists in `script.table()`, the name attribute of the existing script is set to an empty string before it is replaced by the new script.

You must use *scriptVar*`.save()` to save the new script into nonvolatile memory to retain it when the instrument is turned off.

Example 1

```
InstModel = script.new(
   "print(localnode.model)
    print(slot[1].model)
    print(slot[2].model)
    print(slot[3].model)", "InstModel")
InstModel()
```

Creates a new script referenced by the variable `InstModel` with the name `InstModel`.

Runs the script. The model numbers for the mainframe and modules are displayed. Empty models return an error.

Also see

scriptVar.save() (on page 232)

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")
```

name	The name of the script to be restored

Details

This command copies the script from nonvolatile memory into the runtime environment. It also creates a global variable with the same name as the name of the script.

Example

```
script.restore("InstModel")
```

Restore a script named `InstModel` from nonvolatile memory.

Also see

script.delete() (on page 226)

script.table()

This function returns a Lua table 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 index, name in script.table() do name end
```

Details

This function accesses the names of scripts stored in nonvolatile memory, which allows you to process all scripts in nonvolatile memory. The entries are listed randomly.

Each time the body of the function executes, `name` takes on the name of one of the scripts stored in nonvolatile memory. The for loop repeats until all scripts have been iterated.

This function does not return scripts that are in the runtime environment.

Example

```
for index, name in script.table() do
   print(name)
end
```

Retrieve the table listing for scripts and return each script name.

Also see

None

scriptVar.list()

This function generates a script listing.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

scriptVar`.list()`

scriptVar	The name of the 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). The output includes the script control messages `loadscript` and `endscript`.

Example

```
InstModel = script.new("print(localnode.model) print(slot[1].model) print(slot[2].
model) print(slot[3].model)",  "InstModel")
InstModel.list()
```

Creates a script named `InstModel` that returns the model numbers of the mainframe and any installed modules, then confirms the content of the script:

```
loadscript InstModel
print(localnode.model) print(slot[1].model) print(slot[2].model) print(slot[3].mod
el)
endscript
```

Also see

None

scriptVar.name

This attribute contains the name of a script in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not applicable	Not applicable

Usage

scriptVar.name = "*scriptName*"
scriptName = *scriptVar*.name

scriptVar	Name of the variable that references the script
scriptName	A string that represents the name of the script

Details

When setting the script name, this attribute renames the script that the variable *scriptVar* references.

This attribute must be either a valid Lua identifier or the empty string. Changing the name of a script changes the index that is used to access the script in `script.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 is removed from the runtime 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 still reference the script, but the names of the script and variables may not match.

Example

```
ModelNumbers = script.new("print(localnode.model) print(slot[1].model)  print(slo
t[2].model) print(slot[3].model)")
ModelNumbers()
print(ModelNumbers.name)
ModelNumbers.name = "ModelNumbers"
print(ModelNumbers.name)
ModelNumbers.save()
```

This example calls the `script.new()` function to create a script with no name, runs the script, names the script ModelNumbers, and then saves the script in nonvolatile memory.

Also see

script.new() (on page 228)
scriptVar.save() (on page 232)

scriptVar.run()

This function runs a script that is in nonvolatile memory or in the runtime environment.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

scriptVar`.run()`
scriptVar`()`

scriptVar	The name of the variable that references the script

Details

The *scriptVar*`.run()` function runs the script referenced by *scriptVar*. You can also run the script by using *scriptVar()*.

Example

```
test8.run()
```

Runs the script referenced by the variable `test8`.

Also see

None

scriptVar.save()

This function saves a script to nonvolatile memory or to a USB flash drive.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
scriptVar.save()
scriptVar.save("filename")
```

scriptVar	The name of variable that references the script
filename	A string that contains the file name to use when saving the script to a USB flash drive

Details

The *scriptVar*.save() function saves a script to nonvolatile memory or a USB flash drive. The root folder of the USB flash drive has the absolute path /usb1/.

If no *filename* is specified (the file name parameter is an empty string), the script is saved to internal nonvolatile memory. If a *filename* is given, the script is saved to the USB flash drive.

You can add the file extension, but it is not required. The only allowed extension is .tsp.

Example 1

```
InstModel.save()
```

Saves the script referenced by the variable InstModel to nonvolatile memory.

Example 2

```
InstModel.save("/usb1/InstModel2.tsp")
```

Saves the script referenced by the variable InstModel to a file named InstModel2.tsp on your USB flash drive.

Also see

None

scriptVar.source

This attribute contains the source code of a script.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Not applicable	Not saved	Not applicable

Usage

```
code = scriptVar.source
scriptVar.source = nil
```

code	A string that contains the body of the script
scriptVar	The name of the variable that references the script that contains the source code

Details

The `loadscript` and `endscript` keywords are not included in the source code.

The body of the script is a single string with lines separated by the newline character.

The instrument automatically stores the source for all scripts that are loaded on the instrument. To free up memory or to obfuscate the code, assign `nil` to the source attribute of the script. Although this attribute is writable, it can only be set to the `nil` value.

Example

```
SerialNumbersAll = script.new("print(localnode.serialno)  print(slot[1].serialno)
  print(slot[2].serialno)   print(slot[3].serialno)", "SerialNumbersAll" )
print(SerialNumbersAll.source)
```

This example creates a script called `SerialNumbersAll` that returns serial numbers from the mainframe and any installed modules.

Output:

```
print(localnode.serialno)  print(slot[1].serialno)   print(slot[2].serialno)   pri
nt(slot[3].serialno)
```

Also see

scriptVar.list() (on page 229)

---

settime()

This function sets the present time and the real-time clock.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
settime(seconds)
settime(seconds, fractional)
```

seconds	The time in seconds since January 1, 1970, UTC
fractional	The number of fractional seconds

Details

This function sets the date and time of the instrument based on the time parameter (specified in UTC time). UTC time is specified as the number of seconds since Jan 1, 1970, UTC. You can use UTC time from a local time specification, or you can use UTC time from another source (for example, your computer).

Example

```
systemTime = os.time({year = 2025,
month = 7,
day = 31,
hour = 14,
min = 25})
settime(systemTime)
```

Sets the mainframe system date to July 31, 2025, at 2:25 pm.

Also see

os.clock() (on page 218)
os.time() (on page 220)

settimezone()

This function sets the local time zone.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
settimezone(offset)
settimezone("offset", "dstOffset", "dstStart", "dstEnd")
```

offset	String representing offset from UTC
dstOffset	String representing the daylight savings offset from UTC
dstStart	String representing when daylight savings time starts
dstEnd	String representing when daylight savings time ends

Details

You only need to set the time zone if you use the `os.time()` and `os.date()` functions.

If only one parameter is given, the same time offset is used throughout the year. If four parameters are given, time is adjusted twice during the year for daylight savings time.

offset and *dstOffset* are strings of the form `"[+|-]hh[:mm[:ss]]"` that indicate how much time must be added to the local time to get UTC time:

- `hh` is a number between 0 and 23 that represents hours
- `mm` is a number between 0 and 59 that represents minutes
- `ss` is a number between 0 and 59 that represents seconds

The minute, second, +, and − fields are optional.

For example, to set the UTC-5 time zone, you specify the string `"5"`, because UTC-5 is 5 hours behind UTC and you must add 5 hours to the local time to determine UTC time. To specify the time zone UTC4, you specify `"-4"`, because UTC4 is 4 hours ahead of UTC and 4 hours must be subtracted from the local time to determine UTC.

dstStart and *dstEnd* are strings of the form `"MM.w.dw/hh[:mm[:ss]]"` that indicate when daylight savings time begins and ends respectively:

- `MM` is a number between 1 and 12 that represents the month
- `w` is a number between 1 and 5 that represents the week in the month
- `dw` is a number between 0 and 6 that represents the day of the week (where 0 is Sunday)

The rest of the fields represent the time of day that the change takes effect:

- `hh` represents hours
- `mm` represents minutes
- `ss` represents seconds

The minutes and seconds fields are optional.

The week of the month and day of the week fields are not specific dates.

Example

```
settimezone("8", "1", "3.3.0/02", "11.2.0/02")
settimezone(offset)
```

Sets `offset` to equal +8 hours, +1 hour for DST, starts on Mar 14 at 2:00 am, ends on Nov 7 at 2:00 am.

Also see

gettimezone() (on page 176)
os.time() (on page 220)

slot.autorestart

This attribute controls whether or not the instrument automatically restarts modules that have been powered down due to an abnormal condition.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.DISABLE

Usage

```
autorestart = slot.autorestart
slot.autorestart = autorestart
```

autorestart	Automatic restart setting: ▪ Automatically power on and restart modules: `localnode.ENABLE` ▪ Do not automatically power on and restart modules: `localnode.DISABLE`

Details

This attribute controls whether modules that were powered down due to an abnormal condition (such as excessive power supply temperature) restart when the condition is resolved.

Example

```
slot.autorestart = localnode.ENABLE
```

Enables automatic restart of modules that have been powered down due to an abnormal condition.

Also see

slot.autostart (on page 235)
slot.start() (on page 236)
slot.stop() (on page 236)

slot.autostart

This attribute controls whether or not the instrument automatically starts modules that have been inserted.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Not applicable	Nonvolatile memory	localnode.ENABLE

Usage

```
autostart = slot.autostart
slot.autostart = autostart
```

autostart	Automatic start setting: ▪ Automatically power on and start modules: `localnode.ENABLE` ▪ Do not automatically power on and start modules: `localnode.DISABLE`

Details

When autostart is enabled, the instrument automatically powers up and starts modules. This affects modules that are installed when the mainframe is powered on and modules that are inserted when the instrument is running.

The instrument never automatically powers up a module that was stopped by the user.

Example

```
slot.autostart = localnode.ENABLE
```

Enables automatic start of modules that have been inserted

Also see

slot.autorestart (on page 235)
slot.start() (on page 236)
slot.stop() (on page 236)

slot.start()

This function powers on the module in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot.start(slot)`

slot	Module slot number

Details

This command does not wait for the module to finish starting. Use `waitcomplete()` to wait for the module to finish starting.

Example

```
slot.start(1)
waitcomplete()
```

Powers-on the module installed in slot 1 of the mainframe.

Also see

slot.autorestart (on page 235)
slot.autostart (on page 235)
slot.stop() (on page 236)

slot.stop()

This function powers down the module in the specified slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot.stop(Z)`

Z	Module slot number

Details

This command can be used to prepare a module for safe removal while keeping the mainframe powered on.

This command does not wait for the module to finish powering down. Use `waitcomplete()` to wait for the module to finish powering down.

Example

```
slot.stop(1)
waitcomplete()
```

Powers down the module installed in slot 1.

Also see

slot.autorestart (on page 235)
slot.autostart (on page 235)
slot.start() (on page 236)

slot[Z].status.reset()

This function resets all bits in the status model for the specified slot to their default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].status.reset()
```

Z	Module slot number

Details

This function clears all status data structure registers (Enable, Event, NTR, and PTR) to their default values for the specified slot. For information about these registers, refer to Status register set contents (on page 634).

The reset of a slot status model does not reset the mainframe status model.

Example

```
slot[1].status.reset()
```

Resets the status model of the module in slot 1.

Also see

Status model (on page 634)
status.reset() (on page 283)

status.condition

This attribute stores the condition of the status byte register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
statusByte = status.condition
```

statusByte	The status byte; a zero (0) indicates no bits set; other values indicate bit settings

Details

This attribute is used to read the status byte, which is returned as a numeric value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of `1.32000e+02` (which is 132) is read as the value of this register, the binary equivalent is 1000 0100. This value indicates that bits B2 and B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	1	0	0

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred. When an enabled status event occurs, a summary bit is set in this register to indicate the event occurrence.

The individual bits of this register are described in the following table.

Bit	Value and description
B0	Not used.

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set summary bit indicates that an enabled system event has occurred.
	Bit B1 decimal value: 2

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the event log.
	Bit B2 decimal value: 4

B3	Not used.

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the Output Queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Request Service (RQS) or 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 USB serial poll sequence of the MP5103 to obtain the status byte (serial poll byte), B6 is the RQS bit. The set bit indicates that the Request Service (RQS) bit of the status byte (serial poll byte) is set and a service request (SRQ) has occurred.
	▪ When using the `status.condition` register command or the `*STB?` common command to read the status byte, B6 is the MSS bit. Set bit indicates that an enabled summary bit of the status byte register is set.
	Bit B6 decimal value: 64

Bit	Value and description

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

In addition to the above constants, when more than one bit of the register is set, *`statusByte`* equals the sum of their decimal weights.

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 model overview (on page 634)
Status byte and service request (SRQ) (on page 638)

status.node_enable

This attribute stores the system 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 status of the system node enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

This attribute is used to read or write to the system node enable register. Reading the system node enable register returns a value.

Bit	Value and description
B0 and B1	Not used

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the event log.
	Bit B2 decimal value: 4

B3	Not used.

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

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.

As an example, to set the B6 bit of the system node enable register, set `status.node_enable = status.MSS`.

In addition to the above values, *`nodeEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`nodeEnableRegister`* to the sum of their decimal weights. For example, to set bits B2 and B4, set *`nodeEnableRegister`* to 20 (4 + 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)

The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 48 is read as the value of this register, the binary equivalent is 0011 0000. This value indicates that bits B4 and bit B5 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0

* Least significant bit
** Most significant bit

Example 1

```
nodeEnableRegister = status.EAV + status.OSB
status.node_enable = nodeEnableRegister
```

Use constants to set the EAV and OSB bits of the system node enable register.

Example 2

```
-- decimal 192 = binary 11000000
nodeEnableRegister = 192
status.node_enable = nodeEnableRegister
```

Sets the MSB and OSB bits of the system node enable register using a decimal value.

Also see

status.condition (on page 237)
status.system.* (on page 286)
Status byte and service request (SRQ) (on page 638)

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 applicable	0

Usage

```
nodeEventRegister = status.node_event
```

nodeEventRegister	The status of the node event register; a zero (0) indicates no bits set; other values indicate bits are set

Details

This attribute is used to read the status node event register, which is returned as a numeric value (reading this register returns a value). The returned value can indicate that one or more status events occurred.

The returned value can indicate one or more status events occurred.

Bit	Value and description
B0 and B1	Not used

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the event log.
	Bit B2 decimal value: 4

B3	`status.QUESTIONABLE_SUMMARY_BIT`
	`status.QSB`
	Set summary bit indicates that an enabled questionable event has occurred.
	Bit B3 decimal value: 8

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte register is set.
	Bit B6 decimal value: 64

Bit	Value and description

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 48 is read as the value of this register, the binary equivalent is 0011 0000. This value indicates that bits B4 and bit B5 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0

* Least significant bit
** Most significant bit

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:

```
1.30000e+02
```

Converting this output (130) to its binary equivalent yields `1000 0010`. Therefore, this output indicates that the set bits of the status byte condition register are presently B1 (SSB) and B7 (OSB).

Also see

Status byte and service request (SRQ) (on page 638)
status.condition (on page 237)
status.system.* (on page 286)

status.operation.*

These attributes manage the Operation Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not applicable	Not applicable
.enable (RW)	Yes	Status reset	Not applicable	0
.event (R)	Yes	Status reset	Not applicable	0
.ntr (RW)	Yes	Status reset	Not applicable	0
.ptr (RW)	Yes	Status reset	Not applicable	32,704 (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 status of the operation status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate specific bit settings

Details

These attributes read or write the Operation Status register.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `2.04800e+04` (which is 20,480) is read as the value of the condition register, the binary equivalent is 0101 0000 0000 0000. This value indicates that bit B14 (`PROGRAM_RUNNING`) and bit B12 (USER) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B5	Not used
B6	`status.operation.SSTAT` Set bit indicates that the summary bit from the `status.operation.slot.status` register is set. Bit B6 decimal value: 64

Bit	Value and description
B7	`status.operation.INT` Set bit indicates that the interlock is engaged. Bit B7 decimal value: 128
B8	`status.operation.HWD` Set bit indicates that the summary bit of the `status.operation.slot.presence` register is set. Bit B8 decimal value: 256
B9	`status.operation.SLOT` Set bit indicates that the summary bit of the `status.operation.slot.summary` register is set. Bit B9 decimal value: 512
B10	`status.operation.TRGOVR` Set bit indicates that the summary bit from the `status.operation.trigger_overrun` register is set. Bit B10 decimal value: 1,024
B11	`status.operation.REM` Set bit indicates that the summary bit of the `status.operation.remote` register is set. Bit B11 decimal value: 2,048
B12	`status.operation.USER` Set bit indicates that the summary bit from the `status.operation.user` register is set. Bit B12 decimal value: 4,096
B13	`status.operation.INST` Set bit indicates that the summary bit from the `status.operation.instrument` register is set. Bit B13 decimal value: 8,192
B14	`status.operation.PROG` Set bit indicates that a command or program is running, as indicated by the summary bit from the `status.operation.program` register. Bit B14 decimal value: 16,384
B15	Not used

As an example, to set bit B12 of the Operation Status Enable Register, set `status.operation.enable = status.operation.USER`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B12 and B14, set *`operationRegister`* to 20,480 (the sum of 4,096 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.USER + status.operation.PROG
status.operation.enable = operationRegister
```

Uses constants to set the USER and PROG bits of the Operation Status enable register.

Example 2

```
-- decimal 20480 = binary 0101 0000 0000 0000
operationRegister = 20480
status.operation.enable = operationRegister
```

Uses a decimal value to set the USER and PROG bits of the Operation Status enable register.

Also see

Operation Status Registers (on page 658)

status.operation.instrument.*

This attribute contains the operation status instrument summary 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	31,744 (all bits set)

Usage

```
operationRegister = status.operation.instrument.condition
operationRegister = status.operation.instrument.enable
operationRegister = status.operation.instrument.event
operationRegister = status.operation.instrument.ntr
operationRegister = status.operation.instrument.ptr
status.operation.instrument.enable = operationRegister
status.operation.instrument.ntr = operationRegister
status.operation.instrument.ptr = operationRegister
```

operationRegister	The status of the operation event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `2.04800e+04` (which is 20,480) is read as the value of the condition register, the binary equivalent is 0101 0000 0000 0000. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B9	Not used	Not applicable.
B10	`status.operation.instrument.TRGBLND`	Set bit indicates one or more enabled bits for the Operation Status Trigger Blender Summary register is set. Bit B10 decimal value: 1,024

Bit	Value	Description
B11	`status.operation.instrument.TRGTMR`	Set bit indicates one or more enabled bits for the Operation Status Trigger Timer Summary register is set. Bit B11 decimal value: 2,048
B12	`status.operation.instrument.DIGIO`	Set bit indicates one or more enabled bits for the Operation Status Digital I/O Summary register is set. Bit B12 decimal value: 4,096
B13	`status.operation.instrument.TSPLINK`	Set bit indicates one or more enabled bits for the Operation Status TSP-Link Summary register is set. Bit B13 decimal value: 8,192
B14	`status.operation.instrument.LAN`	Set bit indicates one or more enabled bits for the Operation Status LAN Summary register is set. Bit B14 decimal value: 16,384
B15	Not used	Not applicable.

As an example, to set bit B12 of the operation status instrument summary enable register, set `status.operation.instrument.enable = status.operation.instrument.DIGIO`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B12 and B14, set *`operationRegister`* to 20,480 (which is the sum of 4,096 + 16,384).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
-- 4096 = binary 0001 0000 0000 0000
operationRegister = 4096
status.operation.instrument.enable = operationRegister
```

Sets bit B12 of the operation status instrument summary enable register using a decimal value.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)
status.operation.instrument.digio.* (on page 249)
status.operation.instrument.lan.* (on page 255)

status.operation.instrument.digio.*

This attribute contains the operation status digital I/O summary 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	1024 (all bits set)

Usage

```
operationRegister = status.operation.instrument.digio.condition
operationRegister = status.operation.instrument.digio.enable
operationRegister = status.operation.instrument.digio.event
operationRegister = status.operation.instrument.digio.ntr
operationRegister = status.operation.instrument.digio.ptr
status.operation.instrument.digio.enable = operationRegister
status.operation.instrument.digio.ntr = operationRegister
status.operation.instrument.digio.ptr = operationRegister
```

operationRegister	The status of the operation status digital I/O summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status digital I/O summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used
B10	`status.operation.instrument.digio.TRGOVR` Set bit indicates an enabled bit in the Operation Status Digital I/O Overrun Register is set. Bit B10 decimal value: 1,024 Binary value: 0100 0000 0000
B11 to B15	Not used

In addition to the above constant, *operationRegister* can be set to the decimal value of the bit to set.

Example 1

```
status.operation.instrument.digio.enable = status.operation.instrument.digio.TRGOV
R
```

Uses a constant to set the TRGOVR bit of the operation status digital I/O summary enable register.

Example 2

```
status.operation.instrument.digio.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status digital I/O summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.digio.trigger_overrun.* (on page 250)

status.operation.instrument.digio.trigger_overrun.*

This attribute contains the operation status digital I/O trigger overrun register set for lines 1 to 14.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not applicable	Not applicable
.enable (RW)	Yes	Status reset	Not applicable	0
.event (R)	Yes	Status reset	Not applicable	0
.ntr (RW)	Yes	Status reset	Not applicable	0
.ptr (RW)	Yes	Status reset	Not applicable	32,767 (all bits set)

Usage

```
operationRegister = status.operation.instrument.digio.trigger_overrun.condition
operationRegister = status.operation.instrument.digio.trigger_overrun.enable
operationRegister = status.operation.instrument.digio.trigger_overrun.event
operationRegister = status.operation.instrument.digio.trigger_overrun.ntr
operationRegister = status.operation.instrument.digio.trigger_overrun.ptr
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
status.operation.instrument.digio.trigger_overrun.ntr = operationRegister
status.operation.instrument.digio.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status digital I/O overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status digital I/O overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.02600e+03` (which is 1026) is read as the value of the condition register, the binary equivalent is 0000 0100 0000 0010. This value indicates that bit B1 and bit B10 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates that the specified digital I/O line generated an action overrun when it was triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value

B0	`status.operation.instrument.digio.trigger_overrun.EXTENSION_BIT`
	`status.operation.instrument.digio.trigger_overrun.EXT`
	Decimal value: 1

B1	`status.operation.instrument.digio.trigger_overrun.LINE1` Decimal value: 2
B2	`status.operation.instrument.digio.trigger_overrun.LINE2` Decimal value: 4
B3	`status.operation.instrument.digio.trigger_overrun.LINE3` Decimal value: 8
B4	`status.operation.instrument.digio.trigger_overrun.LINE4` Decimal value: 16
B5	`status.operation.instrument.digio.trigger_overrun.LINE5` Decimal value: 32
B6	`status.operation.instrument.digio.trigger_overrun.LINE6` Decimal value: 64
B7	`status.operation.instrument.digio.trigger_overrun.LINE7` Decimal value: 128
B8	`status.operation.instrument.digio.trigger_overrun.LINE8` Decimal value: 256
B9	`status.operation.instrument.digio.trigger_overrun.LINE9` Decimal value: 512
B10	`status.operation.instrument.digio.trigger_overrun.LINE10` Decimal value: 1024
B11	`status.operation.instrument.digio.trigger_overrun.LINE11` Decimal value: 2048
B12	`status.operation.instrument.digio.trigger_overrun.LINE12` Decimal value: 4096
B13	`status.operation.instrument.digio.trigger_overrun.LINE13` Decimal value: 18192
B14	`status.operation.instrument.digio.trigger_overrun.LINE14` Decimal value: 16384

Bit	Value
B15	Not used

As an example, to set bit B1 of the operation status digital I/O overrun enable register, set `status.operation.instrument.digio.trigger_overrun.enable = status.operation.instrument.digio.trigger_overrun.LINE1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.digio.trigger_overrun.LINE1 +
   status.operation.instrument.digio.trigger_overrun.LINE10
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Example 2

```
operationRegister = 1026
status.operation.instrument.digio.trigger_overrun.enable = operationRegister
```

Uses the decimal value 1026 (2+1024) to set bit B1 and bit B10 of the operation status digital I/O overrun enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.digio.* (on page 249)
status.operation.instrument.digio.trigger_overrun2.* (on page 253)
status.operation.trigger_overrun.* (on page 275)

status.operation.instrument.digio.trigger_overrun[2].*

This attribute contains the operation status digital I/O overrun register set for lines 15 through 18.

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	30 (all bits set)

Usage

```
operationRegister = status.operation.instrument.digio.trigger_overrun[2].condition
operationRegister = status.operation.instrument.digio.trigger_overrun[2].enable
operationRegister = status.operation.instrument.digio.trigger_overrun[2].event
operationRegister = status.operation.instrument.digio.trigger_overrun[2].ntr
operationRegister = status.operation.instrument.digio.trigger_overrun[2].ptr
status.operation.instrument.digio.trigger_overrun[2].enable = operationRegister
status.operation.instrument.digio.trigger_overrun[2].ntr = operationRegister
status.operation.instrument.digio.trigger_overrun[2].ptr = operationRegister
```

operationRegister	The status of the operation status digio I/O overrun 2 register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status digital I/O overrun 2 registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B4. For example, if a value of `10` is read as the value of the condition register, the binary equivalent is 1010. This value indicates that bit B1 and bit B4 are set.

B4	B3	B2	B1	B0
>	>	>	>	*
1	0	0	1	0

* Least significant bit

A set bit indicates that the specified digital I/O line generated an action overrun when it was triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.digio.trigger_overrun2.LINE15`	2
B2	`status.operation.instrument.digio.trigger_overrun2.LINE16`	4

Bit	Value	Decimal value
B3	`status.operation.instrument.digio.trigger_overrun2.LINE17`	8
B4	`status.operation.instrument.digio.trigger_overrun2.LINE18`	16
B5 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status digital I/O overrun enable register 2, set `status.operation.instrument.digio.trigger_overrun2.enable = status.operation.instrument.digio.trigger_overrun2.LINE15`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.digio.trigger_overrun[2].LINE15 +
   status.operation.instrument.digio.trigger_overrun[2].LINE17
status.operation.instrument.digio.trigger_overrun[2].enable = operationRegister
```

Uses constants to set bit B1 and bit B3 of the operation status digital I/O overrun enable register 2.

Example 2

```
operationRegister = 18
status.operation.instrument.digio.trigger_overrun[2].enable = operationRegister
```

Uses the decimal value 18 (the sum of 2 + 16) to set bit B1 and bit B4 of the operation status digital I/O overrun enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.digio.* (on page 249)

status.operation.instrument.lan.*

This attribute contains the operation status LAN summary 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	3 (all bits set)

Usage

```
operationRegister = status.operation.instrument.lan.condition
operationRegister = status.operation.instrument.lan.enable
operationRegister = status.operation.instrument.lan.event
operationRegister = status.operation.instrument.lan.ntr
operationRegister = status.operation.instrument.lan.ptr
status.operation.instrument.lan.enable = operationRegister
status.operation.instrument.lan.ntr = operationRegister
status.operation.instrument.lan.ptr = operationRegister
```

operationRegister	The status of the operation status LAN summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings

Details

These attributes are used to read or write to the operation status LAN summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `.00300e+03` (which is 3) is read as the value of the condition register, the binary equivalent is 0000 0000 0000 0011. This value indicates that bit B0 and bit B1 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	0	0	0	0	0	0	1	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	`status.operation.instrument.lan.CON` Set bit indicates that the LAN cable is connected and a link has been detected. Bit B0 decimal value: 1

Bit	Value and description
B1	`status.operation.instrument.lan.CONF` Set bit indicates the LAN is performing its configuration sequence. Bit B1 decimal value: 2
B2 to B15	Not used

As an example, to set bit B0 of the operation status LAN summary enable register, set `status.operation.instrument.lan.enable = status.operation.instrument.lan.CON`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.instrument.lan.CON +
   status.operation.instrument.lan.CONF
status.operation.instrument.lan.enable = operationRegister
```

Use constants to set bit B0 and bit B1 of the operation status LAN summary enable register.

Example 2

```
operationRegister = 3
status.operation.instrument.lan.enable = operationRegister
```

Use the decimal value 3 to set bit B0 and bit B1 of the operation status LAN summary enable register.

Also see

Operation Status Registers (on page 658)

status.operation.instrument.trigger_blender.*

This attribute contains the operation status trigger blender summary 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	1024 (all bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_blender.condition
operationRegister = status.operation.instrument.trigger_blender.enable
operationRegister = status.operation.instrument.trigger_blender.event
operationRegister = status.operation.instrument.trigger_blender.ntr
operationRegister = status.operation.instrument.trigger_blender.ptr
status.operation.instrument.trigger_blender.enable = operationRegister
status.operation.instrument.trigger_blender.ntr = operationRegister
status.operation.instrument.trigger_blender.ptr = operationRegister
```

operationRegister	The status of the operation status trigger blender summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger blender summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents. The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.trigger_blender.TRIGGER_OVERRUN`
	`status.operation.instrument.trigger_blender.TRGOVR`
	Set bit indicates one or more enabled bits for operation status trigger blender overrun register is set.
	Bit B10 decimal value: 1,024
	Binary value:
	0100 0000 0000

B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. For example, to set bit B10, set *operationRegister* to 1024.

Example

```
status.operation.instrument.trigger_blender.enable = 1024
```

Uses a decimal value to set the TRGOVR bit of the operation status trigger blender summary enable.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.trigger_blender.trigger_overrun.* (on page 258)

status.operation.instrument.trigger_blender.trigger_overrun.*

This attribute contains the operation status trigger blender overrun 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	126 (all bits set)

Usage

```
operationRegister =
status.operation.instrument.trigger_blender.trigger_overrun.condition
operationRegister =
status.operation.instrument.trigger_blender.trigger_overrun.enable
operationRegister =
status.operation.instrument.trigger_blender.trigger_overrun.event
operationRegister =
status.operation.instrument.trigger_blender.trigger_overrun.ntr
operationRegister =
status.operation.instrument.trigger_blender.trigger_overrun.ptr
status.operation.instrument.trigger_blender.trigger_overrun.enable =
operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ntr =
operationRegister
status.operation.instrument.trigger_blender.trigger_overrun.ptr =
operationRegister
```

operationRegister	The status of the operation status trigger blender overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings

Details

These attributes are used to read or write to the operation status trigger blender overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `18` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 0010. This value indicates that bit B1 and bit B4 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	0	0	0	1	0	0	1	0

* Least significant bit
** Most significant bit

A set bit value indicates that the specified trigger blender generated an action overrun.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_blender.trigger_overrun.BLND1`	2
B2	`status.operation.instrument.trigger_blender.trigger_overrun.BLND2`	4
B3	`status.operation.instrument.trigger_blender.trigger_overrun.BLND3`	8
B4	`status.operation.instrument.trigger_blender.trigger_overrun.BLND4`	16
B5 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger blender overrun enable register, set `status.operation.instrument.trigger_blender.trigger_overrun.enable = status.operation.instrument.trigger_blender.trigger_overrun.BLND1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B4, set *`operationRegister`* to 18 (which is the sum of 2 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
status.operation.instrument.trigger_blender.trigger_overrun.enable
   = status.operation.instrument.trigger_blender.trigger_overrun.BLND1
```

Uses the constant to set the bit for blender 1 of the operation status trigger blender overrun enable register.

Example

```
status.operation.instrument.trigger_blender.trigger_overrun.enable = 18
```

Uses the decimal value to set the bits for blenders 1 and 4 of the operation status trigger blender overrun enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.trigger_blender.* (on page 257)

status.operation.instrument.trigger_timer.*

This attribute contains the operation status trigger timer summary register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not applicable	Not applicable
.enable (RW)	Yes	Status reset	Not applicable	0
.event (R)	Yes	Status reset	Not applicable	0
.ntr (RW)	Yes	Status reset	Not applicable	0
.ptr (RW)	Yes	Status reset	Not applicable	1024 (all bits set)

Usage

```
operationRegister = status.operation.instrument.trigger_timer.condition
operationRegister = status.operation.instrument.trigger_timer.enable
operationRegister = status.operation.instrument.trigger_timer.event
operationRegister = status.operation.instrument.trigger_timer.ntr
operationRegister = status.operation.instrument.trigger_timer.ptr
status.operation.instrument.trigger_timer.enable = operationRegister
status.operation.instrument.trigger_timer.ntr = operationRegister
status.operation.instrument.trigger_timer.ptr = operationRegister
```

operationRegister	The status of the operation status trigger timer summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status trigger timer summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.trigger_timer.TRIGGER_OVERRUN`
	`status.operation.instrument.trigger_timer.TRGOVR`
	
	Set bit indicates one or more enabled bits for the operation status trigger timer overrun register is set.
	
	Bit B10 decimal value: 1,024
	
	Binary value:
	0100 0000 0000

Bit	Value and description
B11 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set.

Example

```
status.operation.instrument.trigger_timer.enable = 1024
```

Uses the decimal value to set the TRGOVR bit of the operation status trigger timer summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.trigger_timer.trigger_overrun.* (on page 261)

status.operation.instrument.trigger_timer.trigger_overrun.*

This attribute contains the operation status trigger timer overrun 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	510 (all bits set)

Usage

```
operationRegister =
status.operation.instrument.trigger_timer.trigger_overrun.condition
operationRegister =
status.operation.instrument.trigger_timer.trigger_overrun.enable
operationRegister =
status.operation.instrument.trigger_timer.trigger_overrun.event
operationRegister =
status.operation.instrument.trigger_timer.trigger_overrun.ntr
operationRegister =
status.operation.instrument.trigger_timer.trigger_overrun.ptr
status.operation.instrument.trigger_timer.trigger_overrun.enable =
operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ntr =
operationRegister
status.operation.instrument.trigger_timer.trigger_overrun.ptr =
operationRegister
```

operationRegister	The status of the operation status trigger timer trigger overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status trigger timer overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates the specified timer generated an action overrun because it was still processing a delay from a previous trigger when a new trigger was received.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.trigger_timer.trigger_overrun.TMR1`	2
B2	`status.operation.instrument.trigger_timer.trigger_overrun.TMR2`	4
B3	`status.operation.instrument.trigger_timer.trigger_overrun.TMR3`	8
B4	`status.operation.instrument.trigger_timer.trigger_overrun.TMR4`	16
B5	`status.operation.instrument.trigger_timer.trigger_overrun.TMR5`	32
B6	`status.operation.instrument.trigger_timer.trigger_overrun.TMR6`	64
B7	`status.operation.instrument.trigger_timer.trigger_overrun.TMR7`	128
B8	`status.operation.instrument.trigger_timer.trigger_overrun.TMR8`	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the operation status trigger timer trigger overrun enable register, set `status.operation.instrument.trigger_timer.trigger_overrun.enable = status.operation.instrument.trigger_timer.trigger_overrun.TMR1`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights.

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**

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	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example 1

```
status.operation.instrument.trigger_timer.trigger_overrun.enable =
   status.operation.instrument.trigger_timer.trigger_overrun.TMR3
```

Uses a constant to set the timer 3 bit of the operation status trigger timer overrun enable register.

Example 2

```
status.operation.instrument.trigger_timer.trigger_overrun.enable = 18
```

Uses a constant to set timer bits B1 and B4 (decimal values 2 and 16) of the operation status trigger timer overrun enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.trigger_timer.* (on page 260)

status.operation.instrument.tsplink.*

This attribute contains the operation status TSP-Link summary 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	1024 (all bits set)

Usage

```
operationRegister = status.operation.instrument.tsplink.condition
operationRegister = status.operation.instrument.tsplink.enable
operationRegister = status.operation.instrument.tsplink.event
operationRegister = status.operation.instrument.tsplink.ntr
operationRegister = status.operation.instrument.tsplink.ptr
status.operation.instrument.tsplink.enable = operationRegister
status.operation.instrument.tsplink.ntr = operationRegister
status.operation.instrument.tsplink.ptr = operationRegister
```

operationRegister	The status of the operation status TSP-Link summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); the only valid value other than 0 is 1024

Details

These attributes are used to read or write to the operation status TSP-Link summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B9	Not used

B10	`status.operation.instrument.tsplink.TRIGGER_OVERRUN`
	`status.operation.instrument.tsplink.TRGOVR`
	Set bit indicates one or more enabled bits for the operation status TSP-Link overrun register is set.
	Bit B10 decimal value: 1,024
	Binary value:
	0100 0000 0000

B11 to B15	Not used

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set.

Example

```
status.operation.instrument.tsplink.enable = 1024
```

Uses the decimal value to set the trigger overrun bit (B10) of the operation status TSP-Link summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.tsplink.trigger_overrun.* (on page 265)

status.operation.instrument.tsplink.trigger_overrun.*

This attribute contains the operation status TSP-Link overrun 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	14 (all bits set)

Usage

```
operationRegister =
status.operation.instrument.tsplink.trigger_overrun.condition
operationRegister = status.operation.instrument.tsplink.trigger_overrun.enable
operationRegister = status.operation.instrument.tsplink.trigger_overrun.event
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ntr
operationRegister = status.operation.instrument.tsplink.trigger_overrun.ptr
status.operation.instrument.tsplink.trigger_overrun.enable = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ntr = operationRegister
status.operation.instrument.tsplink.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status TSP-Link overrun register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status TSP-Link overrun registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `10` is read as the value of the condition register, the binary equivalent is 0000 0000 0000 1010. This value indicates that bit B1 and bit B3 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	0	0	0	0	1	0	1	0

* Least significant bit
** Most significant bit

A set bit indicates that the specified line generated an action overrun when triggered to generate an output trigger.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.operation.instrument.tsplink.trigger_overrun.LINE1`	2
B2	`status.operation.instrument.tsplink.trigger_overrun.LINE2`	4

Bit	Value	Decimal value
B3	`status.operation.instrument.tsplink.trigger_overrun.LINE3`	8
B4 to B15	Not used	Not applicable

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B3, set *`operationRegister`* to 10 (which is the sum of 2 + 8).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
status.operation.instrument.tsplink.trigger_overrun.enable =
   status.operation.instrument.tsplink.trigger_overrun.LINE1
```

Uses a constant to set the line 1 bit of the operation status TSP-Link overrun enable register.

Example 2

```
status.operation.instrument.tsplink.trigger_overrun.enable = 10
```

Uses the decimal value to set bits for lines 1 and 3 of the operation status TSP-Link overrun enable register.

Also see

Operation Status Registers (on page 658)
status.operation.instrument.trigger_timer.* (on page 260)

status.operation.program.*

This attribute contains the Operation Status Program Status register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	2 (valid bit set)

Usage

```
operationRegister = status.operation.program.condition
operationRegister = status.operation.program.enable
operationRegister = status.operation.program.event
operationRegister = status.operation.program.ntr
operationRegister = status.operation.program.ptr
status.operation.program.enable = operationRegister
status.operation.program.ntr = operationRegister
status.operation.program.ptr = operationRegister
```

operationRegister	The status of the program status register; a zero (0) indicates no bits set (also send 0 to clear all bits)

Details

These attributes are used to read or write to the Program Status register. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.program.PROG` Set bit indicates there is a program running Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2 to B15	Reserved

In addition to the above constants, *operationRegister* can be set to the decimal value of the bit to set.

Example 1

```
status.operation.program = status.operation.program.PROG
```

Uses a constant to set the PROG bit, B1, of the Operation Status Program Status register.

Example 2

```
status.operation.program = 2
```

Uses the decimal value to set bit B1 of the Operation Status Program Status register.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

status.operation.remote.*

This attribute contains the operation status remote summary 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	2050 (all bits set)

Usage

```
operationRegister = status.operation.remote.condition
operationRegister = status.operation.remote.enable
operationRegister = status.operation.remote.event
operationRegister = status.operation.remote.ntr
operationRegister = status.operation.remote.ptr
status.operation.remote.enable = operationRegister
status.operation.remote.ntr = operationRegister
status.operation.remote.ptr = operationRegister
```

operationRegister	The status of the operation status remote summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate set bits

Details

These attributes are used to read or write to the operation status remote summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.remote.CAV` Set bit indicates there is a command available in the execution queue. Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010

Bit	Value and description
B2 to B10	Not used
B11	`status.operation.remote.PRMPT` Set bit indicates command prompts are enabled. Bit B11 decimal value: 2,048 Binary value: 0000 0100 0000 0000
B12 to B15	Not used

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values.

Example 1

```
status.operation.remote.enable = status.operation.remote.CAV
```

Uses a constant to set the CAV bit, B1, of the operation status remote summary enable register.

Example 2

```
status.operation.remote.enable = 2050
```

Uses the decimal value (the sum of 2 + 2,048) to set bits B1 and B11 of the Operation Status Remote Summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

status.operation.slot.presence.*

This attribute contains the operation status hardware presence 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	14 (all bits set)

Usage

```
operationRegister = status.operation.slot.presence.condition
operationRegister = status.operation.slot.presence.enable
operationRegister = status.operation.slot.presence.event
operationRegister = status.operation.slot.presence.ntr
operationRegister = status.operation.slot.presence.ptr
status.operation.slot.presence.enable = operationRegister
status.operation.slot.presence.ntr = operationRegister
status.operation.slot.presence.ptr = operationRegister
```

operationRegister	The status of the operation status hardware presence register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status hardware presence summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.slot.presence.SLOT1` Set bit indicates there is a module in slot 1 Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2	`status.operation.slot.presence.SLOT2` Set bit indicates there is a module in slot 2 Bit B2 decimal value: 4 Binary value: Binary value: 0000 0000 0000 0100

Bit	Value and description
B3	`status.operation.slot.presence.SLOT3` Set bit indicates there is a module in slot 3 Bit B3 decimal value: 8 Binary value: Binary value: 0000 0000 1000
B4 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal values. For example, to set bits B1 and B3, set *operationRegister* to 10 (which is the sum of 2 + 8).

Example 1

```
status.operation.slot.presence = status.operation.slot.presence.SLOT1
```

Uses a constant to set the SLOT1 bit, B1, of the operation status hardware presence register.

Example 2

```
status.operation.slot.presence = 2
```

Uses the decimal value to set bit B1 of the operation status hardware presence register.

Also see

Operation Status Registers (on page 658)
status.operation (on page 244)

status.operation.slot.status.*

This attribute contains the operation Slot Status register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	14 (all bits set)

Usage

```
operationRegister = status.operation.slot.status.condition
operationRegister = status.operation.slot.status.enable
operationRegister = status.operation.slot.status.event
operationRegister = status.operation.slot.status.ntr
operationRegister = status.operation.slot.status.ptr
status.operation.slot.status.enable = operationRegister
status.operation.slot.status.ntr = operationRegister
status.operation.slot.status.ptr = operationRegister
```

operationRegister	The status of the operation status slot.status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation Slot Status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.slot.status.SLOT1` Set bit indicates the measurement, questionable, or operation bit on slot 1 is set. Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2	`status.operation.slot.status.SLOT2` Set bit indicates the measurement, questionable, or operation bit on slot 2 is set. Bit B2 decimal value: 4 Binary value: Binary value: 0000 0000 0000 0100
B3	`status.operation.slot.status.SLOT3` Set bit indicates the measurement, questionable, or operation bit on slot 3 is set. Bit B3 decimal value: 8 Binary value: Binary value: 0000 0000 1000
B4 to B15	Not used

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values. For example, to set bits B1 and B3, set *`operationRegister`* to 10 (which is the sum of 2 + 8).

Example 1

```
status.operation.slot.status.enable = status.operation.slot.status.SLOT1
```

Uses a constant to set the SLOT1 bit B1 of the operation status slot status register.

Example 2

```
status.operation.slot.status.enable = 2
```

Uses the decimal value to set bit B1 of the operation status slot status register.

Also see

Operation Status Registers (on page 658)
status.operation (on page 244)

status.operation.slot.summary.*

This attribute contains the Slot Summary 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	14 (all bits set)

Usage

```
operationRegister = status.operation.slot.summary.condition
operationRegister = status.operation.slot.summary.enable
operationRegister = status.operation.slot.summary.event
operationRegister = status.operation.slot.summary.ntr
operationRegister = status.operation.slot.summary.ptr
status.operation.slot.summary.enable = operationRegister
status.operation.slot.summary.ntr = operationRegister
status.operation.slot.summary.ptr = operationRegister
```

operationRegister	The status of the operation status slot summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the Slot Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

These bits are from the Slot Status Register of the device in the slot and indicate if the measurement, questionable, or operation bit on the respective module is set

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`status.operation.slot.summary.SLOT1` Set bit indicates that the operation summary of slot 1 is set Bit B1 decimal value: 2 Binary value: 0000 0000 0000 0010
B2	`status.operation.slot.presence.SLOT2` Set bit indicates that the operation summary of slot 2 is set Bit B2 decimal value: 4 Binary value: Binary value: 0000 0000 0000 0100

Bit	Value and description
B3	`status.operation.slot.presence.SLOT3` Set bit indicates that the operation summary of slot 3 is set Bit B3 decimal value: 8 Binary value: Binary value: 0000 0000 1000
B4 to B15	Not used

As an example, to set bit B1 of the operation status slot summary register, set `status.operation.slot.summary = status.operation.slot.summary.SLOT1`.

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values. For example, to set bits B1 and B3, set *`operationRegister`* to 10 (which is the sum of 2 + 8).

Example 1

```
status.operation.slot.summary = status.operation.slot.summary.SLOT1
```

Uses a constant to set the SLOT1 bit, B1, of the operation status slot summary register.

Example 2

```
status.operation.slot.summary = 2
```

Uses the decimal value to set bit B1 of the operation status slot summary register.

Also see

Operation Status Registers (on page 658)
status.operation (on page 244)

status.operation.trigger_overrun.*

This attribute contains the operation status trigger overrun summary 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	15,360 (all bits set)

Usage

```
operationRegister = status.operation.trigger_overrun.condition
operationRegister = status.operation.trigger_overrun.enable
operationRegister = status.operation.trigger_overrun.event
operationRegister = status.operation.trigger_overrun.ntr
operationRegister = status.operation.trigger_overrun.ptr
status.operation.trigger_overrun.enable = operationRegister
status.operation.trigger_overrun.ntr = operationRegister
status.operation.trigger_overrun.ptr = operationRegister
```

operationRegister	The status of the operation status trigger overrun summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings

Details

These attributes are used to read or write to the operation status trigger overrun summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 2048 is read as the value of the condition register, the binary equivalent is 0000 1000 0000 0000. This value indicates that bit B11 is set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

The bits in this register summarize events in other registers. A set bit in this summary register indicates that an enabled event in one of the summarized registers is set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0 to B10	Not used

Bit	Value
B10	`status.operation.trigger_overrun.TRGBLD` Set bit indicates one of the enabled bits in the Operation Status Trigger Timer Overrun register is set. Bit B12 decimal value: 1,024
B11	`status.operation.trigger_overrun.TRGTMR` Set bit indicates one of the enabled bits in the Operation Status Trigger Timer Overrun register is set. Bit B11 decimal value: 2,048
B12	`status.operation.trigger_overrun.DIGIO` Set bit indicates one of the enabled bits in the Operation Status Digital I/O Overrun Event register is set. Bit B12 decimal value: 4,096
B13	`status.operation.trigger_overrun.TSPLINK` Set bit indicates one of the enabled bits in the Operation Status TSP-Link Overrun Event register is set. Bit B13 decimal value: 8,192
B14	Not used
B15	Not used

As an example, to set bit B12 of the operation status trigger overrun summary enable register, set `status.operation.trigger_overrun.enable = status.operation.trigger_overrun.DIGIO`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B12 and B13, set *`operationRegister`* to 12,288 (the sum of 4096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
operationRegister = status.operation.trigger_overrun.DIGIO + status.operation.trig
ger_overrun.TSPLINK
status.operation.trigger_overrun.enable = operationRegister
```

Uses constants to set bit B12 and bit B13 of the operation status trigger overrun summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation (on page 244)

status.operation.user.*

These attributes manage the operation status user register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (RW)	Yes	Status reset	Not saved	0
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	32,767 (all bits set)

Usage

```
operationRegister = status.operation.user.condition
operationRegister = status.operation.user.enable
operationRegister = status.operation.user.event
operationRegister = status.operation.user.ntr
operationRegister = status.operation.user.ptr
status.operation.user.condition = operationRegister
status.operation.user.enable = operationRegister
status.operation.user.ntr = operationRegister
status.operation.user.ptr = operationRegister
```

operationRegister	The status of the operation status user register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status user registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bits B0 and B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.operation.user.BIT0`	1
B1	`status.operation.user.BIT1`	2

Bit	Value	Decimal value
B2	`status.operation.user.BIT2`	4
B3	`status.operation.user.BIT3`	8
B4	`status.operation.user.BIT4`	16
B5	`status.operation.user.BIT5`	32
B6	`status.operation.user.BIT6`	64
B7	`status.operation.user.BIT7`	128
B8	`status.operation.user.BIT8`	256
B9	`status.operation.user.BIT9`	512
B10	`status.operation.user.BIT10`	1,024
B11	`status.operation.user.BIT11`	2,048
B12	`status.operation.user.BIT12`	4,096
B13	`status.operation.user.BIT13`	8,192
B14	`status.operation.user.BIT14`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the operation status user enable register, set `status.operation.user.enable = status.operation.user.BIT0`.

In addition to the above constants, *`operationRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = status.operation.user.BIT11 + status.operation.user.BIT14
status.operation.user.enable = operationRegister
```

Uses constants to set bits B11 and B14 of the operation status user enable register.

Example 2

```
-- 18432 = binary 0100 1000 0000 0000
operationRegister = 18432
status.operation.enable = operationRegister
```

Uses the decimal value 18,432 (the sum of 2048 + 16,384) to set bits B11 and B14 of the operation status user enable register.

Also see

status.operation.* (on page 244)

---

status.request_enable

This attribute stores the service request (SRQ) enable register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Status reset	Not saved	0

Usage

```
requestSRQEnableRegister = status.request_enable
status.request_enable = requestSRQEnableRegister
```

requestSRQEnableRegister	The status of the service request (SRQ) enable register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

This attribute is used to read or write to the SRQ enable register. Reading the SRQ enable register returns a value.

The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 48 is read as the value of this register, the binary equivalent is 0011 0000. This value indicates that bits B4 and bit B5 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
0	0	1	1	0	0	0	0

* Least significant bit
** Most significant bit

The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used

Bit	Value and description

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set system summary bit indicates that an enabled event in the System Summary register occurred.
	Bit B0 decimal value: 2
	Not used

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the event log.
	Bit B2 decimal value: 4

B3	Not used.

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled standard event has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled operation event has occurred.
	Bit B7 decimal value: 128

In addition to the above values, *`requestSRQEnableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`requestSRQEnableRegister`* to the sum of their decimal weights. For example, to set bits B1 and B7, set *`requestSRQEnableRegister`* to 130 (2 + 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.SSB + status.OSB
status.request_enable = requestSRQEnableRegister
```

Uses constants to set the SSB and OSB bits of the service request (SRQ) enable register.

Example 2

```
-- decimal 130 = binary 10000010
requestSRQEnableRegister = 130
status.request_enable = requestSRQEnableRegister
```

Uses a decimal value to set the SSB and OSB bits of the service request (SRQ) enable register.

Example 3

```
status.request_enable = status.SSB
```

Uses the constant to set the SSB bit of the service request (SRQ) enable register.

Also see

Status byte and service request (SRQ) (on page 638)
status.condition (on page 237)
status.system.* (on page 286)

status.request_event

This attribute stores the service request (SRQ) event register.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	0

Usage

requestSRQEventRegister = status.request_event

requestSRQEventRegister	The status of the service request event register; a zero (0) indicates no bits set; other values indicate bits are set

Details

This attribute is used to read the service request event register, which is returned as a numeric value. Reading this register returns a value. The binary equivalent of the value of this attribute indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B7. For example, if a value of 1.32000e+02 (which is 132) is read as the value of this register, the binary equivalent is 1000 0100. This value indicates that bits B2 and B7 are set.

B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	*
1	0	0	0	0	1	0	0

* Least significant bit
** Most significant bit

The returned value can indicate one or more status events occurred.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used

B1	`status.SYSTEM_SUMMARY_BIT`
	`status.SSB`
	Set summary bit indicates that an enabled event in the System Summary Register has occurred.
	Bit B1 decimal value: 2

B2	`status.ERROR_AVAILABLE`
	`status.EAV`
	Set summary bit indicates that an error or status message is present in the event log.
	Bit B2 decimal value: 4

B3	Not used

B4	`status.MESSAGE_AVAILABLE`
	`status.MAV`
	Set summary bit indicates that a response message is present in the output queue.
	Bit B4 decimal value: 16

B5	`status.EVENT_SUMMARY_BIT`
	`status.ESB`
	Set summary bit indicates that an enabled event in the Standard Event Status Register has occurred.
	Bit B5 decimal value: 32

B6	`status.MASTER_SUMMARY_STATUS`
	`status.MSS`
	Set bit indicates that an enabled Master Summary Status (MSS) bit of the Status Byte Register is set.
	Bit B6 decimal value: 64

B7	`status.OPERATION_SUMMARY_BIT`
	`status.OSB`
	Set summary bit indicates that an enabled event in the Operation Status Register has occurred.
	Bit B7 decimal value: 128

In addition to the above constants, *`requestEventRegister`* can be set to the decimal equivalent of the bits set. When more than one bit of the register is set, *`requestEventRegister`* contains the sum of their decimal weights.

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

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example

```
requestEventRegister = status.request_event
print(requestEventRegister)
```

Reads the status request event register.

Sample output:

```
1.32000e+02
```

Converting this output (132) to its binary equivalent yields 1000 0100.

Therefore, this output indicates that the set bits of the status request event register are presently B2 (EAV) and B7 (OSB).

Also see

status.condition (on page 237)
status.system.* (on page 286)
Status byte and service request (SRQ) (on page 638)

status.reset()

This function resets all bits in the mainframe status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
status.reset()
```

Details

This function clears all status data structure registers (Enable, Event, NTR, and PTR) to their default values. For information about these registers, refer to Status register set contents (on page 634).

This command does not affect the status model of the installed modules.

Example

```
status.reset()
```

Resets the status model of the mainframe.

Also see

Status model (on page 634)
slot[Z].status.reset() (on page 237)

status.standard.*

These attributes manage the Standard Event Status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	189 (all bits set)

Usage

```
standardRegister = status.standard.condition
standardRegister = status.standard.enable
standardRegister = status.standard.event
standardRegister = status.standard.ntr
standardRegister = status.standard.ptr
status.standard.enable = standardRegister
status.standard.ntr = standardRegister
status.standard.ptr = standardRegister
```

standardRegister	The status of the standard event status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings

Details

These attributes are used to read or write to the standard event status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value

B0	`status.standard.OPERATION_COMPLETE`
	`status.standard.OPC`
	
	Set bit indicates that all pending selected instrument operations are completed and the instrument is ready to accept new commands. The bit is set in response to an `*OPC` command. The `opc()` function can be used in place of the `*OPC` command.
	
	Bit B0 decimal value: 1

Bit	Value
B1	Not used

B2	`status.standard.QUERY_ERROR`
	`status.standard.QYE`
	Set bit indicates that you attempted to read data from an empty Output Queue.
	Bit B2 decimal value: 4

B3	`status.standard.DEVICE_DEPENDENT_ERROR`
	`status.standard.DDE`
	Set bit indicates that an instrument operation did not execute properly due to some internal condition.
	Bit B3 decimal value: 8

B4	`status.standard.EXECUTION_ERROR`
	`status.standard.EXE`
	Set bit indicates that the instrument detected an error while trying to execute a command.
	Bit B4 decimal value: 16

B5	`status.standard.COMMAND_ERROR`
	`status.standard.CME`
	Set bit indicates that a command error has occurred. Command errors include semantic errors, such as a command that was misspelled.
	Bit B5 decimal value: 32

B6	Not used

B7	`status.standard.POWER_ON`
	`status.standard.PON`
	Set bit indicates that the instrument has been turned off and turned back on since the last time this register has been read.
	Bit B7 decimal value: 128

B8 to B15	Not used

As an example, to set bit B0 of the standard event status enable register, set `status.standard.enable = status.standard.OPC`.

In addition to the above constants, *`standardRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`standardRegister`* to the sum of their decimal weights. For example, to set bits B0 and B4, set *`standardRegister`* to 17 (which is the sum of 1 + 16).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)

Example 1

```
standardRegister = status.standard.OPC + status.standard.EXE
status.standard.enable = standardRegister
```

Uses constants to set the OPC and EXE bits of the standard event status enable register.

Example 2

```
-- decimal 17 = binary 0001 0001
standardRegister = 17
status.standard.enable = standardRegister
```

Uses the decimal value to set the OPC and EXE bits of the standard event status enable register.

Also see

Standard Event Register (on page 635)

status.system.*

These attributes manage the TSP-Link™ system summary register of the status model 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	32,767 (all bits set)

Usage

```
enableRegister = status.system.condition
enableRegister = status.system.enable
enableRegister = status.system.event
enableRegister = status.system.ntr
enableRegister = status.system.ptr
status.system.enable = enableRegister
status.system.ntr = enableRegister
status.system.ptr = enableRegister
```

enableRegister	The status of the system summary register; a zero (0) indicates no bits set; other values indicate bit settings

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system.EXTENSION_BIT` `status.system.EXT`	1
B1	`status.system.NODE1`	2
B2	`status.system.NODE2`	4
B3	`status.system.NODE3`	8
B4	`status.system.NODE4`	16
B5	`status.system.NODE5`	32
B6	`status.system.NODE6`	64
B7	`status.system.NODE7`	128
B8	`status.system.NODE8`	256
B9	`status.system.NODE9`	512
B10	`status.system.NODE10`	1,024
B11	`status.system.NODE11`	2,048
B12	`status.system.NODE12`	4,096
B13	`status.system.NODE13`	8,192
B14	`status.system.NODE14`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary status enable register, set `status.system.enable = status.system.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values.

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	B7	B6	B5	B4	B3	B2	B1	B0
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example 1

```
enableRegister = status.system.NODE11 + status.system.NODE14
status.system.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the System Summary enable register. In this example, `enableRegister` is set to 18,432 (which is the sum of 2,048 + 16,384).

Also see

status.system2.* (on page 289)
System summary registers (on page 643)

status.system2.*

These attributes manage the TSP-Link system summary register of the status model 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	32,767 (all bits set)

Usage

```
enableRegister = status.system2.condition
enableRegister = status.system2.enable
enableRegister = status.system2.event
enableRegister = status.system2.ntr
enableRegister = status.system2.ptr
status.system2.enable = enableRegister
status.system2.ntr = enableRegister
status.system2.ptr = enableRegister
```

enableRegister	The status of the system summary 2 register; a zero (0) indicates no bits set; other values indicate bits are set

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system2.EXTENSION_BIT` `status.system2.EXT`	1
B1	`status.system2.NODE15`	2
B2	`status.system2.NODE16`	4
B3	`status.system2.NODE17`	8

Bit	Value	Decimal value
B4	`status.system2.NODE18`	16
B5	`status.system2.NODE19`	32
B6	`status.system2.NODE20`	64
B7	`status.system2.NODE21`	128
B8	`status.system2.NODE22`	256
B9	`status.system2.NODE23`	512
B10	`status.system2.NODE24`	1,024
B11	`status.system2.NODE25`	2,048
B12	`status.system2.NODE26`	4,096
B13	`status.system2.NODE27`	8,192
B14	`status.system2.NODE28`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 2 enable register, set `status.system2.enable = status.system2.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system2.NODE25 + status.system2.NODE28
status.system2.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 2 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system2.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 2 enable register.

Also see

status.system.* (on page 286)
status.system3.* (on page 291)

status.system3.*

These attributes manage the TSP-Link system summary register of the status model 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	32,767 (all bits set)

Usage

```
enableRegister = status.system3.condition
enableRegister = status.system3.enable
enableRegister = status.system3.event
enableRegister = status.system3.ntr
enableRegister = status.system3.ptr
status.system3.enable = enableRegister
status.system3.ntr = enableRegister
status.system3.ptr = enableRegister
```

enableRegister	The status of the system summary 3 register; a zero (0) indicates no bits set; other values indicate bits are set

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system3.EXTENSION_BIT` `status.system3.EXT`	1
B1	`status.system3.NODE29`	2
B2	`status.system3.NODE30`	4
B3	`status.system3.NODE31`	8
B4	`status.system3.NODE32`	16
B5	`status.system3.NODE33`	32
B6	`status.system3.NODE34`	64
B7	`status.system3.NODE35`	128
B8	`status.system3.NODE36`	256
B9	`status.system3.NODE37`	512
B10	`status.system3.NODE38`	1,024
B11	`status.system3.NODE39`	2,048
B12	`status.system3.NODE40`	4,096
B13	`status.system3.NODE41`	8,192
B14	`status.system3.NODE42`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 3 enable register, set `status.system3.enable = status.system3.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example 1

```
enableRegister = status.system3.NODE39 + status.system3.NODE42
status.system3.enable = enableRegister
```

Uses constants to set bits B11 and B14 of the system summary 3 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system3.enable = enableRegister
```

Uses the decimal value to set bits B11 and B14 of the system summary 3 enable register.

Also see

status.system2.* (on page 289)
status.system4.* (on page 293)

status.system4.*

These attributes manage the TSP-Link> system summary register of the status model 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	32,767 (all bits set)

Usage

```
enableRegister = status.system4.condition
enableRegister = status.system4.enable
enableRegister = status.system4.event
enableRegister = status.system4.ntr
enableRegister = status.system4.ptr
status.system4.enable = enableRegister
status.system4.ntr = enableRegister
status.system4.ptr = enableRegister
```

enableRegister	The status of the system summary 4 register; a zero (0) indicates no bits set; other values indicate bits are set

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.29000e+02` (which is 129) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	`status.system4.EXTENSION_BIT` `status.system4.EXT`	1
B1	`status.system4.NODE43`	2
B2	`status.system4.NODE44`	4
B3	`status.system4.NODE45`	8
B4	`status.system4.NODE46`	16
B5	`status.system4.NODE47`	32
B6	`status.system4.NODE48`	64
B7	`status.system4.NODE49`	128
B8	`status.system4.NODE50`	256
B9	`status.system4.NODE51`	512
B10	`status.system4.NODE52`	1,024
B11	`status.system4.NODE53`	2,048
B12	`status.system4.NODE54`	4,096
B13	`status.system4.NODE55`	8,192
B14	`status.system4.NODE56`	16,384
B15	Not used	Not applicable

As an example, to set bit B0 of the system summary 4 enable register, set `status.system4.enable = status.system4.enable.EXT`.

In addition to the above constants, *`enableRegister`* can be set to the decimal value of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal values.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system4.NODE53 + status.system4.NODE56
status.system2.enable = enableRegister
```

Uses constants to set bit B11 and bit B14 of the system summary 4 enable register.

Example 2

```
-- decimal 18432 = binary 0100 1000 0000 0000
enableRegister = 18432
status.system4.enable = enableRegister
```

Uses a decimal value to set bit B11 and bit B14 of the system summary 4 enable register. 18,432 is the sum of 2,048 + 16,384.

Also see

status.system3.* (on page 291)
status.system5.* (on page 296)

status.system5.*

These attributes manage the TSP-Link™ system summary register of the status model 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 (all bits set)

Usage

```
enableRegister = status.system5.condition
enableRegister = status.system5.enable
enableRegister = status.system5.event
enableRegister = status.system5.ntr
enableRegister = status.system5.ptr
status.system5.enable = enableRegister
status.system5.ntr = enableRegister
status.system5.ptr = enableRegister
```

enableRegister	The status of the system summary 5 register; a zero (0) indicates no bits set; other values indicate bits are set

Details

In an expanded system (TSP-Link), these attributes are used to read or write to the system summary registers. They are set using a constant or a numeric value, but are returned as a numeric value. The binary equivalent of the value indicates which register bits are set. In the binary equivalent, the least significant bit is bit B0, and the most significant bit is bit B15. For example, if a value of `1.30000e+02` (which is 130) is read as the value of the condition register, the binary equivalent is 0000 0000 1000 0010. This value indicates that bit B1 and bit B7 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents and Enable and transition registers. The individual bits of this register are defined in the following table.

Bit	Value	Decimal value
B0	Not used	Not applicable
B1	`status.system5.NODE57`	2
B2	`status.system5.NODE58`	4
B3	`status.system5.NODE59`	8

Bit	Value	Decimal value
B4	`status.system5.NODE60`	16
B5	`status.system5.NODE61`	32
B6	`status.system5.NODE62`	64
B7	`status.system5.NODE63`	128
B8	`status.system5.NODE64`	256
B9 to B15	Not used	Not applicable

As an example, to set bit B1 of the system summary 5 enable register, set `status.system5.enable = status.system5.NODE57`.

In addition to the above constants, *`enableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`enableRegister`* to the sum of their decimal weights.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
enableRegister = status.system5.NODE57 + status.system5.NODE60
status.system2.enable = enableRegister
```

Uses constants to set bits B1 and B4 of the system summary 5 enable register.

Example 2

```
-- decimal 18 = binary 0000 0000 0001 0010
enableRegister = 18
status.system5.enable = enableRegister
```

Uses the decimal value 18 (the sum of 2+16) to set bits B1 and B4 of the system summary 5 enable register.

Also see

status.system4.* (on page 293)

timer.cleartime()

This function resets the timer to zero (0) seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
timer.cleartime()
```

Details

This function resets the timer.

Example

```
timer.cleartime()
-- (intervening code)
time = timer.readtime()
print(time)
```

Resets the timer and then measures the time since the reset.

Output:

```
1.469077e+01
```

The above output indicates that `timer.readtime()` was executed 14.69077 seconds after `timer.cleartime()`.

Also see

timer.readtime() (on page 298)

timer.readtime()

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.readtime()
```

time	The elapsed time in seconds (1 µs resolution)

Example

```
timer.cleartime()
-- (intervening code)
time = timer.readtime()
print(time)
```

Resets the timer and then measures the time since the reset.

Output:

```
1.469077e+01
```

The above output indicates that `timer.readtime()` was executed 14.69077 seconds after `timer.cleartime()`.

Also see

timer.cleartime() (on page 298)

trigger.blender[N].clear()

This function clears the blender event detector and resets the overrun indicator of blender *N*.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].clear()
```

N	The blender number: 1 to 4

Details

This command sets the blender event detector to the undetected state and resets the overrun indicator of the event detector. This command will also reset the associated Operation Status Trigger Blender Overrun Summary Register (on page 657).

Example

```
trigger.blender[2].clear()
```

Clears the event detector for blender 2.

Also see

trigger.blender[N].overrun (on page 300)

trigger.blender[N].EVENT_ID

This constant contains the trigger blender event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.blender[N].EVENT_ID
```

eventID	Trigger event number
N	The blender number: 1 to 4

Details

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this trigger blender.

Example

```
digio.trigger[1].stimulus = trigger.blender[2].EVENT_ID
```

Set the trigger stimulus of digital I/O trigger 1 to be controlled by the trigger blender 2 event.

Also see

None

trigger.blender[N].orenable

This attribute selects whether the blender performs OR operations or AND operations

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Trigger blender *N* reset	Not applicable	false (AND mode)

Usage

```
orenable = trigger.blender[N].orenable
trigger.blender[N].orenable = orenable
```

orenable	The type of operation: ▪ OR operation: `true` ▪ AND operation: `false`
N	The blender number: 1 to 4

Details

This command selects whether the blender waits for any one event (`OR`) or waits for all selected events (`AND`) before signaling an output event.

Example

```
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].reset() (on page 301)

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	Trigger blender *N* clear Trigger blender *N* reset	Not applicable	Not applicable

Usage

```
overrun = trigger.blender[N].overrun
```

overrun	Trigger blender overrun state: `true` or `false`
N	The blender number: 1 to 4

Details

Indicates if an event was ignored because the event detector for this blender was already in the detected state when the event occurred. This is an indication of the state of the event detector that is built into the event blender itself.

This command does not indicate if an overrun occurred in any other part of the trigger model or in any other trigger object that is monitoring the event. It also is not an indication of an action overrun.

Example

```
print(trigger.blender[1].overrun)
```

If an event was ignored, the output is `true`.

If an event was not ignored, the output is `false`.

Also see

Event overruns (on page 85)
trigger.blender[N].clear() (on page 299)
trigger.blender[N].reset() (on page 301)

trigger.blender[N].reset()

This function resets some of the trigger blender settings to the default values.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.blender[N].reset()
```

N	The trigger event blender: 1 to 4

Details

The `trigger.blender[N].reset()` function resets the following attributes to their factory defaults:

- `trigger.blender[N].orenable` (`true`)
- `trigger.blender[N].stimulus[M]` (`trigger.EVENT_NONE`)

This command also clears `trigger.blender[N].overrun`, and also resets the associated Operation Status Trigger Blender Overrun Summary Register (on page 657).

Example

```
trigger.blender[1].reset()
```

Resets the trigger blender 1 settings to factory defaults.

Also see

Operation Status Trigger Blender Overrun Summary Register. (on page 657)
trigger.blender[N].orenable (on page 300)
trigger.blender[N].overrun (on page 300)
trigger.blender[N].stimulus[M] (on page 302)

trigger.blender[N].stimulus[M]

This attribute specifies the events that trigger the blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Trigger blender N reset	Not applicable	trigger.EVENT_NONE

Usage

```
eventID = trigger.blender[N].stimulus[M]
trigger.blender[N].stimulus[M] = eventID
```

eventID	The event that triggers the blender action; see **Details**
N	An integer representing the trigger event blender (1 to 4)
M	An integer representing the stimulus index (1 to 4)

Details

There are four stimulus inputs that can each select a different event.

Use zero or `trigger.EVENT_NONE` to disable the blender input.

The *eventID* parameter may be one of the existing trigger event IDs shown in the following table.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
digio.trigger[3].mode = digio.MODE_TRIGGER_OUT
digio.trigger[3].mode = digio.EDGE_RISING
digio.trigger[5].mode = digio.MODE_TRIGGER_OUT
digio.trigger[5].mode = digio.EDGE_RISING
trigger.blender[1].orenable = true
trigger.blender[1].stimulus[1] = digio.trigger[3].EVENT_ID
trigger.blender[1].stimulus[2] = digio.trigger[5].EVENT_ID
```

Generate a trigger blender 1 event when a digital I/O trigger happens on line 3 or 5.

Also see

trigger.blender[N].orenable (on page 300)
trigger.blender[N].reset (on page 301)

trigger.blender[N].wait()

This function waits for a blender trigger event to occur on a specific blender.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.blender[*N*].wait(*timeout*)

triggered	Trigger detection indication for blender
N	The trigger blender on which to wait: 1 to 4
timeout	Maximum amount of time in seconds to wait for the trigger blender event

Details

This function waits for an event blender trigger event. If one or more trigger events were detected since the last time `trigger.blender[N].wait()` or `trigger.blender[N].clear()` was called, this function returns immediately.

After detecting a trigger with this function, the event detector automatically resets and rearms. This is true regardless of the number of events detected.

Example

```
digio.trigger[3].mode = digio.MODE_TRIGGER_IN
digio.trigger[5].mode = digio.MODE_TRIGGER_IN
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 in 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 output. If the trigger event has not happened, then `false` is output after the timeout expires.

Also see

trigger.blender[N].clear() (on page 299)

trigger.clear()

This function clears the command interface trigger event detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
trigger.clear()
```

Details

The trigger event detector indicates if a trigger event has been detected since the last `trigger.wait()` call. The `trigger.clear()` function clears the trigger event detector and discards the history of command interface trigger events.

Also see

trigger.wait() (on page 314)

trigger.detector[N].clear()

This command clears the trigger event detector and the overrun indicator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
trigger.detector[N].clear()
```

N	The detector number: 1 to 8

Details

The event detector of a trigger enters the detected state when an event is detected. It is cleared when `trigger.detector[N].clear()` is called.

This command clears the event detector of the specified index, discards the history of the detector, and clears the `trigger.detector[N].overrun` attribute.

Example

```
trigger.detector[2].clear()
```

Clears the events on detector 2.

Also see

trigger.detector[N].overrun (on page 305)

trigger.detector[N].overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	
Attribute (R)	Yes	Power cycle Mainframe reset	Not applicable	Not applicable

Usage

```
overrun = trigger.detector[N].overrun
```

overrun	Trigger overrun state: `true` or `false`
N	The detector number: 1 to 8

Details

This attribute indicates if an event was ignored because the event detector was already in the detected state when an event occurred.

Example

```
print(trigger.detector[1].overrun)
```

If an event was ignored, the output is `true`. If an event was not ignored, the output is `false`.

Also see

trigger.detector[N].clear() (on page 304)

trigger.detector[N].stimulus

This attribute specifies which event triggers the detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset	Not saved	trigger.EVENT_NONE

Usage

```
eventID = trigger.detector[N].stimulus
trigger.detector[N].stimulus = eventID
```

eventID	The event that triggers the detector
N	The detector number: 1 to 8

Details

The *eventID* parameter can be the trigger event of any trigger event on the mainframe or any installed modules.

The event IDs that can be used for this command include mainframe-specific event IDs as well as event IDs for any installed modules capable of generating a trigger event. See the following table.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18

ID	Description
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
trigger.detector[1].stimulus = digio.trigger[1].EVENT_ID
digio.trigger[1].assert()
```

Generates an event on the detector when triggering event occurs on line 1.

Also see

trigger.detector[N].wait() (on page 306)

trigger.detector[N].wait()

This command waits for an event to be detected by the detector.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.detector[*N*].wait(*timeout*)

triggered	Trigger event detection indicator: ▪ Not detected: false ▪ Detected: true
N	The detector number: 1 to 8
timeout	The 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.detector[N].wait() or trigger.detector[N].clear() was called, this function immediately returns true. Otherwise, it will wait for up to the number of sections specified in *timeout* for the event to occur. It returns true if the event occurs and false if the event does not occur before the *timeout* value.

After waiting for a trigger event with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
trigger.detector[1].stimulus=digio.trigger[1].EVENT_ID
digio.trigger[1].assert()
print(trigger.detector[1].wait(3))
```

Generates an event on trigger detector 1 when a digital I/O trigger occurs on line 1.

Waits 3 s while checking if an event occurred on trigger detector 1.

Also see

trigger.detector[N].clear() (on page 304)
trigger.detector[N].stimulus (on page 305)

trigger.EVENT_NONE

This trigger event ID is used to disconnect stimulus inputs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = trigger.EVENT_NONE

eventID	The trigger event ID

Details

This event ID is never generated. It can be used to disconnect a stimulus event that was set to another trigger event to disconnect it from all events.

Example

```
trigger.detector[1].stimulus = trigger.EVENT_NONE
```

No trigger event on detector 1.

Also see

None

trigger.generator[N].assert()

This function generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

trigger.generator[*N*].assert()

N	The generator number: 1 to 8

Details

Use this function to directly trigger events from the command interface or a script. For example, you can trigger a sweep while the instrument is under script control.

Example

```
trigger.generator[2].assert()
```

Generates a trigger event on generator 2.

Also see

trigger.generator[N].EVENT_ID (on page 308)

trigger.generator[N].EVENT_ID

This constant identifies the trigger event generated by the trigger event generator.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventID = trigger.generator[*N*].EVENT_ID

eventID	The trigger event number
N	The generator number: 1 to 8

Details

This constant is a number that identifies events generated by this generator.

To have another trigger object respond to trigger events generated by this generator, set the stimulus of the other object to the value of this constant.

Example

```
digio.trigger[5].stimulus = trigger.generator[2].EVENT_ID
```

Uses a trigger event on generator 2 to be the stimulus for digital I/O trigger line 5.

Also see

trigger.generator[N].assert() (on page 307)

trigger.timer[N].clear()

This function clears the timer event detector and overrun indicator for the specified trigger timer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

trigger.timer[*N*].clear()

N	Trigger timer number: 1 to 8

Details

This command sets the timer event detector to the undetected state and resets the overrun indicator.

Example

```
trigger.timer[1].clear()
```

Clears trigger timer 1.

Also see

trigger.timer[N].count (on page 309)

trigger.timer[N].count

This attribute sets the number of events to generate each time the timer generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Mainframe reset trigger.timer[N].reset	Not saved	1

Usage

```
count = trigger.timer[N].count
trigger.timer[N].count = count
```

count	Number of times to repeat the trigger: 0 to 1,048,575
N	Trigger timer number: 1 to 8

Details

If the count is set to a number greater than 1, the timer automatically starts the next trigger timer delay at the expiration of the previous delay.

Set the count to zero (0) to cause the timer to generate trigger events indefinitely.

If you use the trigger timer with a trigger model, make sure the count value is the same or more than any count values expected in the trigger model.

Example

```
print(trigger.timer[1].count)
```

Read trigger count for timer number 1.

Also see

trigger.timer[N].clear() (on page 308)
trigger.timer[N].delay (on page 310)
trigger.timer[N].reset() (on page 312)

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	Mainframe reset trigger.timer[N].reset	Not saved	10e−6 (10 µs)

Usage

```
interval = trigger.timer[N].delay
trigger.timer[N].delay = interval
```

interval	Delay interval in seconds: 5e−7 s to 400 s
N	Trigger timer number: 1 to 8

When the timer is enabled, the timer uses this delay each time it is triggered.

Example

```
trigger.timer[1].delay = 50e-6
```

Set the trigger timer 1 to delay for 50 µs.

Also see

trigger.timer[N].reset() (on page 312)

trigger.timer[N].EVENT_ID

This constant specifies the trigger timer event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = trigger.timer[N].EVENT_ID
```

eventID	The trigger event number
N	Trigger timer number: 1 to 8

Details

This constant is an identification number that identifies events generated by this timer.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to events from this timer.

Example

```
trigger.timer[1].stimulus = digio.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the digital I/O trigger 2 event.

Also see

None

trigger.timer[N].overrun

This attribute indicates if an event was ignored because of the event detector state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Mainframe reset trigger.timer[N].clear trigger.timer[N].reset	Not applicable	false

Usage

```
overrun = trigger.timer[N].overrun
```

overrun	Trigger overrun state: `true` or `false`
N	Trigger timer number: 1 to 8

Details

This command indicates if an event was ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the timer itself. It does not indicate if an overrun occurred in any other part of the trigger model or in any other construct that is monitoring the delay completion event. It also is not an indication of a delay overrun.

Delay overrun indications are provided in the status model.

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 312)

trigger.timer[N].passthrough

This attribute enables or disables the timer trigger pass-through mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Mainframe reset trigger.timer[N].reset	Not saved	false (disabled)

Usage

```
passthrough = trigger.timer[N].passthrough
trigger.timer[N].passthrough = passthrough
```

passthrough	The state of pass-through mode; set to one of the following values: ▪ Enabled: `true` ▪ Disabled: `false`
N	Trigger timer number: 1 to 8

Details

When pass-through mode is enabled, triggers are passed through immediately and initiate the delay. When disabled, a trigger only initiates a delay.

Example

```
trigger.timer[1].passthrough = true
```

Enables pass-through mode on trigger timer 1.

Also see

trigger.timer[N].reset() (on page 312)

trigger.timer[N].reset()

This function resets the trigger timer settings to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`trigger.timer[N].reset()`

N	Trigger timer number: 1 to 8

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].passthrough`
- `trigger.timer[N].stimulus`

It also clears `trigger.timer[N].overrun`.

Example

```
trigger.timer[1].reset()
```

Resets the attributes associated with timer 1 to factory default values.

Also see

trigger.timer[N].count (on page 309)
trigger.timer[N].delay (on page 310)
trigger.timer[N].overrun (on page 311)
trigger.timer[N].passthrough (on page 311)
trigger.timer[N].stimulus (on page 313)

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	Mainframe reset trigger.timer[N].reset	Not saved	0

Usage

```
eventID = trigger.timer[N].stimulus
trigger.timer[N].stimulus = eventID
```

eventID	The event that triggers the timer delay
N	Trigger timer number: 1 to 8

Details

The event IDs that can be used for this command include mainframe-specific event IDs and event IDs for any installed modules capable of generating a trigger event. See the following table.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
print(trigger.timer[1].stimulus)
```

Prints the event that starts a trigger 1 timer action.

Also see

trigger.timer[N].reset (on page 312)

trigger.timer[N].wait()

This function waits for a trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.timer[*N*].wait(*timeout*)

triggered	Trigger detection indication
N	Trigger timer number: 1 to 8
timeout	Maximum amount of time in seconds to wait for the trigger

Details

If one or more trigger events were detected since the last time `trigger.timer[N].wait()` or `trigger.timer[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = trigger.timer[3].wait(10)
print(triggered)
```

Waits up to 10 s for a trigger on timer 3.

If `false` is returned, no trigger was detected during the 10 s timeout.

If `true` is returned, a trigger was detected.

Also see

trigger.timer[N].clear() (on page 308)

trigger.wait()

This function waits for a command interface trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

triggered = trigger.wait(*timeout*)

triggered	A trigger was detected during the timeout period: `true` No triggers were detected during the timeout period: `false`
timeout	Maximum amount of time in seconds to wait for the trigger

Details

This function waits up to *timeout* seconds for a trigger on the active command interface. A command interface trigger occurs when:

- A USBTMC TRIGGER message is received (USB only)
- A *TRG message is received

If one or more of these trigger events were previously detected, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
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 period.

If true is returned, a trigger was detected.

Also see

*TRG (on page 632)
trigger.clear() (on page 304)

tsplink.group

This attribute contains the group number of a TSP-Link node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
groupNumber = tsplink.group
tsplink.group = groupNumber
```

groupNumber	The group number of the TSP-Link node: 0 to 64

Details

To remove the node from all groups, set the attribute value to 0.

When the node is turned off, the group number for that node changes to 0.

The master node can be assigned to any group. You can also include other nodes in the group that includes the master. Any nodes that are set to 0 are automatically included in the group that contains the master node, regardless of the group that is assigned to the master node.

Example

```
tsplink.group = 3
```

Assign the instrument to TSP-Link group number 3.

Also see

None

tsplink.initialize()

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.initialize()
nodesFound = tsplink.initialize(expectedNodes)
```

nodesFound	The number of nodes actually found on the system
expectedNodes	The number of nodes expected on the system: 1 to 64

Details

This function erases all information regarding other nodes connected on the TSP-Link system and regenerates the system configuration. This function must be called at least once before any remote nodes can be accessed. If the node number for any instrument is changed, the TSP-Link must be reset again.

If *expectedNodes* is not given, this function generates an error if no other nodes are found on the TSP-Link network.

If *nodesFound* is less than *expectedNodes*, an error is generated. Note that the node on which the command is running is counted as a node. For example, giving an expected node count of 1 does not generate any errors, even if there are no other nodes on the TSP-Link network.

Also returns the number of nodes found.

This command replaces `tsplink.reset()`, which is deprecated.

Example

```
nodesFound = tsplink.initialize(2)
print("Nodes found = " .. nodesFound)
```

Perform a TSP-Link initialization and indicate how many nodes are found.

Sample output if two nodes are found:

```
Nodes found = 2
```

Sample output if fewer nodes are found and if `localnode.showerrors = 1`:

```
1219, TSP-Link found fewer nodes than expected
Nodes found = 1
```

Also see

localnode.showerrors (on page 213)
tsplink.node (on page 317)
tsplink.state (on page 319)

tsplink.master

This attribute reads the node number assigned to the master node.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
masterNodeNumber = tsplink.master
```

masterNodeNumber	The node number of the master node

Details

After doing a TSP-Link reset (`tsplink.initialize()`), use this attribute to access the node number of the master in a set of instruments connected over TSP-Link.

Example

```
LinkMaster = tsplink.master
```

Store the TSP-Link master node number in a variable called `LinkMaster`.

Also see

tsplink.initialize() (on page 315)

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	1

Usage

```
nodeNumber = tsplink.node
tsplink.node = nodeNumber
```

nodeNumber	The node number of the instrument or enclosure: 1 to 64

Details

This command sets the TSP-Link node number and saves the value in nonvolatile memory.

Changes to the node number do not take effect until `tsplink.initialize()` is executed on any node in the system.

Each link connected to the TSP-Link system must be assigned a different node number.

Example

```
tsplink.node = 3
```

Sets the TSP-Link node for this instrument to number 3.

Also see

tsplink.initialize() (on page 315)
tsplink.state (on page 319)

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)
```

data	The state of the synchronization line ▪ Low: 0 ▪ High: 1
N	The trigger line: 1 to 3

Example

```
data = tsplink.readbit(3)
print(data)
```

Assume line 3 is set high and is then read.

Output:

```
1.000000e+00
```

Also see

tsplink.readport() (on page 318)
tsplink.writebit() (on page 330)

tsplink.readport()

This function reads the TSP-Link trigger lines as a digital I/O port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
data = tsplink.readport()
```

data	Numeric value that indicates which lines are set

Details

The binary equivalent of the returned value indicates the input pattern on the I/O port. The least significant bit of the binary number corresponds to line 1 and the value of bit 3 corresponds to line 3. For example, a returned value of 2 has a binary equivalent of 010. This indicates that line 2 is high (1), and that the other two lines are low (0).

Example

```
data = tsplink.readport()
print(data)
```

Reads state of all three TSP-Link lines.

Assuming line 2 is set high, the output is:

```
2.000000e+00
```

(binary 010)

The format of the output may vary depending on the ASCII precision setting.

Also see

tsplink.readbit() (on page 318)
tsplink.writebit() (on page 330)

tsplink.state

This attribute describes the TSP-Link online state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
state = tsplink.state
```

state	TSP-Link state: `online` or `offline`

Details

When the mainframe power is first turned on, the state is `offline`. After `tsplink.initialize()` is successful, the state is `online`.

Example

```
state = tsplink.state
print(state)
```

Read the state of the TSP-Link system. If it is online, the output is:

```
online
```

Also see

tsplink.initialize() (on page 315)
tsplink.node (on page 317)

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()
```

N	The trigger line: 1 to 3

Details

The set pulse width determines how long the trigger is asserted.

Example

```
tsplink.trigger[2].assert()
```

Asserts trigger on trigger line 2.

Also see

tsplink.trigger[N].clear() (on page 320)
tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].overrun (on page 325)
tsplink.trigger[N].pulsewidth (on page 325)
tsplink.trigger[N].release() (on page 326)
tsplink.trigger[N].stimulus (on page 327)
tsplink.trigger[N].wait() (on page 329)

tsplink.trigger[N].clear()

This function clears the event detector for a TSP-Link trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tsplink.trigger[N].clear()
```

N	The trigger line: 1 to 3

Details

The trigger event detector enters the detected state when an event is detected. `tsplink.trigger[N].clear()` clears a trigger event detector, discards the history of the trigger line, and clears the `tsplink.trigger[N].overrun` attribute.

Example

```
tsplink.trigger[2].clear()
```

Clears trigger event on synchronization line 2.

Also see

tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].overrun (on page 325)
tsplink.trigger[N].release() (on page 326)
tsplink.trigger[N].stimulus (on page 327)
tsplink.trigger[N].wait() (on page 329)

tsplink.trigger[N].edge

This command configures the edge detection mode for a TSP-Link I/O trigger.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset tsplink.trigger[N].reset	Not applicable	tsplink.EDGE_FALLING

Usage

```
triggerEdge = tsplink.trigger[N].edge
tsplink.trigger[N].edge = triggerEdge
```

triggerEdge	The trigger edge selection; refer to **Details** for values
N	The trigger line: 1 to 3

Details

This command is used for input trigger detection. To control output trigger generation, use the `tsplink.trigger[N].logic` command.

To directly control the line state, set the mode of the line to digital and use the write command. When the digital line mode is set for open drain, the edge settings assert a TTL low-pulse.

Set *triggerEdge* to one of the values shown in the following table.

***triggerEdge* value**	**Description**
tsplink.EDGE_FALLING	Detects falling-edge triggers as input when the line is configured as an input or open drain
tsplink.EDGE_RISING	Detects rising-edge triggers as input when the line is configured as an open drain
tsplink.EDGE_EITHER	Detects rising- or falling-edge triggers as input when the line is configured as an input or open drain

Example

```
tsplink.trigger[3].edge = tsplink.EDGE_RISING
```

Sets the trigger edge detection mode for TSP-Link I/O line 3 to detect rising edges as input triggers.

Also see

tsplink.trigger[N].logic (on page 322)

tsplink.trigger[N].EVENT_ID

This constant identifies the number that is used for a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventID = tsplink.trigger[N].EVENT_ID
```

eventID	The trigger event number
N	The trigger line: 1 to 3

Details

This number is used by the TSP-Link trigger line when it detects an input trigger.

Set the stimulus of any trigger object to the value of this constant to have the trigger object respond to trigger events from this line.

Example

```
trigger.timer[1].stimulus = tsplink.trigger[2].EVENT_ID
```

Sets the trigger stimulus of trigger timer 1 to the TSP-Link trigger 2 event.

Also see

None

tsplink.trigger[N].logic

This attribute sets the output logic of the trigger event generator to positive or negative for the specified line.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset tsplink.trigger[N].reset()	Not applicable	tsplink.LOGIC_NEGATIVE

Usage

```
triggerLogic = tsplink.trigger[N].logic
tsplink.trigger[N].logic = triggerLogic
```

triggerLogic	The trigger mode: ▪ Assert a TTL-high pulse for output: `tsplink.LOGIC_POSITIVE` ▪ Assert a TTL-low pulse for output: `tsplink.LOGIC_NEGATIVE`
N	TSP-Link trigger line: 1 to 3

Details

This command use used for output triggers. To control input trigger detection, use the `tsplink.trigger[N].edge` command.

Only use this attribute with lines that are configured for trigger modes.

Example

```
tsplink.trigger[3].mode = tsplink.MODE_TRIGGER_OUT
tsplink.trigger[3].logic = tsplink.LOGIC_POSITIVE
```

Sets TSP-Link line 3 mode to be a trigger output and sets the output logic of the trigger event generator to positive.

Also see

tsplink.trigger[N].clear() (on page 320)
tsplink.trigger[N].edge (on page 321)
tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].reset (on page 326)

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	Power cycle Mainframe reset tsplink.trigger[N].reset	Not saved	tsplink.MODE_DIGITAL_IN

Usage

```
triggerMode = tsplink.trigger[N].mode
tsplink.trigger[N].mode = triggerMode
```

triggerMode	The trigger mode; see **Details** for values
N	The trigger line: 1 to 3

Details

This attribute controls the mode in which the trigger event detector and the output trigger generator operate on the given trigger line.

Set *triggerMode* to one of the values shown in the following table.

***triggerMode* value**	**Description**
tsplink.MODE_DIGITAL_IN	Digital control, input
tsplink.MODE_DIGITAL_OUT	Digital control, output
tsplink.MODE_DIGITAL_OPEN_DRAIN	Digital control, open drain
tsplink.MODE_TRIGGER_IN	Trigger control, input
tsplink.MODE_TRIGGER_OUT	Trigger control, output
tsplink.MODE_TRIGGER_OPEN_DRAIN	Trigger control, open drain
tsplink.MODE_SYNCHRONOUS_MASTER	Synchronous master
tsplink.MODE_SYNCHRONOUS_ACCEPTOR	Synchronous acceptor

You can use this command to place each TSP-Link line into one of the following modes:

- Digital open-drain, output, or input
- Trigger open-drain, output, or input
- Trigger synchronous master or synchronous acceptor

A digital line allows direct control of the digital I/O lines by writing a bit pattern to the lines. A trigger line uses the digital I/O lines to detect triggers.

The following settings of *triggerMode* set the line for direct control as a digital line:

- `tsplink.MODE_DIGITAL_IN`: The instrument automatically detects externally generated logic levels. You can read an input line, but you cannot write to it.
- `tsplink.MODE_DIGITAL_OUT`: You can set the line as logic high (+5 V) or as logic low (0 V). The default level is logic low (0 V). When the instrument is in output mode, the line is actively driven high or low.
- `tsplink.MODE_DIGITAL_OPEN_DRAIN`: Configures the line to be an open-drain signal. The line can serve as an input, an output or both. When a digital I/O line is used as an input in open-drain mode, you must write a `1` to it.

The following settings of *triggerMode* set the line for direct control as a trigger line:

- `tsplink.MODE_TRIGGER_IN`: The line automatically responds to and detects externally generated triggers. It detects falling-edge, rising-edge, or either-edge triggers as input. This line state uses the edge setting specified by the `tsplink.trigger[N].edge` attribute.
- `tsplink.MODE_TRIGGER_OUT`: The line is automatically set high or low depending on the output logic specified by the `tsplink.trigger[N].logic` attribute. Use the negative logic setting when you want to generate a falling edge trigger and use the positive logic setting when you want to generate a rising edge trigger.
- `tsplink.MODE_TRIGGER_OPEN_DRAIN`: Configures the line to be an open-drain signal. You can use the line to detect input triggers or generate output triggers. This line state uses the edge setting specified by the `tsplink.trigger[N].edge` attribute and the logic setting specified by the `tsplink.trigger[N].logic` attribute.

When the line is set as a synchronous acceptor, the line detects the falling-edge input triggers and automatically latches and drives the trigger line low. Asserting an output trigger releases the latched line.

When the line is set as a synchronous master, the line detects rising-edge triggers as input. For output, the line asserts a TTL-low pulse.

Example

```
tsplink.trigger[1].mode = tsplink.MODE_DIGITAL_OUTPUT
node[10].tsplink.trigger[1].mode = tsplink.MODE_DIGITAL_INPUT
-- Set bit B1 high on output line.
tsplink.writebit(1, 1)
-- Read bit B1 on the input line.
data = node[10].tsplink.readbit(1)
print(data)
```

The following programming example illustrates how to set bit B1 of the TSP-Link digital I/O port high and then read the entire port value. The output is similar to:

```
1
```

Also see

digio.trigger[N].edge (on page 145)
digio.writebit() (on page 155)
digio.writeport() (on page 155)
tsplink.trigger[N].assert() (on page 319)
tsplink.trigger[N].clear() (on page 320)
tsplink.trigger[N].overrun (on page 325)
tsplink.trigger[N].release() (on page 326)
tsplink.trigger[N].reset() (on page 326)
tsplink.trigger[N].stimulus (on page 327)
tsplink.trigger[N].wait() (on page 329)

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 (RW)	Yes	Power cycle Mainframe reset tsplink.trigger[N].clear tsplink.trigger[N].reset	Not applicable	Not applicable

Usage

```
overrun = tsplink.trigger[N].overrun
```

overrun	Trigger overrun state: `true` or `false`
N	The trigger line: 1 to 3

Details

This command indicates whether an event has been ignored because the event detector was already in the detected state when the event occurred.

This is an indication of the state of the event detector built into the synchronization line itself.

It does not indicate if an overrun occurred in any other part of the trigger model, or in any other construct that is monitoring the event. It also is not an indication of an output trigger overrun. Output trigger overrun indications are provided in the status model.

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 319)
tsplink.trigger[N].clear() (on page 320)
tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].release() (on page 326)
tsplink.trigger[N].reset() (on page 326)
tsplink.trigger[N].stimulus (on page 327)
tsplink.trigger[N].wait() (on page 329)

tsplink.trigger[N].pulsewidth

This attribute sets the length of time that the trigger line is asserted for output triggers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset tsplink.trigger[N].reset	Not saved	10e-6 (10 µs)

Usage

```
width = tsplink.trigger[N].pulsewidth
tsplink.trigger[N].pulsewidth = width
```

width	The pulse width, up to 400 s; set to 0 to trigger indefinitely
N	The trigger line: 1 to 3

Details

To assert the pulse width indefinitely, set the pulse width to 0 seconds.

Example

```
tsplink.trigger[3].pulsewidth = 20e-6
```

Sets pulse width for trigger line 3 to 20 µs.

Also see

tsplink.trigger[N].release() (on page 326)

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()`

N	The trigger line: 1 to 3

Details

Releases a trigger that was asserted with an indefinite pulse width. It also releases a trigger that was latched in response to receiving a synchronous mode trigger.

Example

```
tsplink.trigger[3].release()
```

Releases trigger line 3.

Also see

tsplink.trigger[N].assert() (on page 319)
tsplink.trigger[N].clear() (on page 320)
tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].overrun (on page 325)
tsplink.trigger[N].pulsewidth (on page 325)
tsplink.trigger[N].stimulus (on page 327)
tsplink.trigger[N].wait() (on page 329)

tsplink.trigger[N].reset()

This function resets some of the TSP-Link trigger attributes to their factory defaults.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`tsplink.trigger[N].reset()`

N	The trigger line: 1 to 3

Details

The `tsplink.trigger[N].reset()` function resets the following attributes to their factory defaults:

- `tsplink.trigger[N].mode`
- `tsplink.trigger[N].stimulus`
- `tsplink.trigger[N].pulsewidth`

This also clears `tsplink.trigger[N].overrun`.

Example

```
tsplink.trigger[3].reset()
```

Resets TSP-Link trigger line 3 attributes to factory default values.

Also see

tsplink.trigger[N].mode (on page 323)
tsplink.trigger[N].overrun (on page 325)
tsplink.trigger[N].pulsewidth (on page 325)
tsplink.trigger[N].stimulus (on page 327)

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	Power cycle Mainframe reset tsplink.trigger[N].reset	Nonvolatile memory	1

Usage

```
eventID = tsplink.trigger[N].stimulus
tsplink.trigger[N].stimulus = eventID
```

eventID	The event identifier for the triggering event
N	The trigger line: 1 to 3

Details

To disable automatic trigger assertion on the synchronization line, set this attribute to zero (0).

Do not use this attribute when triggering under script control. Use `tsplink.trigger[N].assert()` instead.

The event IDs that can be used for this command include mainframe-specific event IDs as well as event IDs for any installed modules capable of generating a trigger event. See the following table.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4

ID	Description
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
-- Alias SMU
local smu = slot[1].smu[1]
local triggerModel = slot[1].trigger.model
-- Set up TSP-Link trigger line.
tsplink.trigger[1].stimulus = triggerModel.EVENT_NOTIFY1
tsplink.trigger[1].logic = tsplink.LOGIC_POSITIVE
-- Initiate buffers for trigger model use.
smu.trigger.measure.iv(smu.defbuffer1, smu.defbuffer2)
-- Create a trigger model to make measurements and
-- initiate TSP-Link trigger events.
triggerModel.create("tm")
triggerModel.addblock.measure("tm", "measure", 1)
triggerModel.addblock.notify("tm", "notify-tsplink", triggerModel.EVENT_NOTIFY1)
triggerModel.addblock.delay.constant("tm", "delay", 100e-3)
triggerModel.addblock.branch.counter("tm", "branch", "measure", 5)
triggerModel.initiate("tm")
```

Creates a trigger model that uses the TSP-Link stimulus.

Also see

None

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*)

triggered	Trigger detection indication; set to one of the following values: ▪ A trigger is detected during the timeout period: `true` ▪ A trigger is not detected during the timeout period: `false`
N	The trigger line: 1 to 3
timeout	The timeout value in seconds

Details

This function waits up to the timeout value for an input trigger. If one or more trigger events were detected since the last time `tsplink.trigger[N].wait()` or `tsplink.trigger[N].clear()` was called, this function returns immediately.

After waiting for a trigger with this function, the event detector is automatically reset and rearmed. This is true regardless of the number of events detected.

Example

```
triggered = tsplink.trigger[3].wait(10)
print(triggered)
```

Waits up to 10 seconds for a trigger on TSP-Link line 3.

If `false` is returned, no trigger was detected during the 10-second timeout.

If `true` is returned, a trigger was detected.

Also see

tsplink.trigger[N].clear() (on page 320)

tsplink.writebit()

This function sets a TSP-Link trigger line high or low.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`tsplink.writebit(N, data)`

N	The trigger line: 1 to 3
data	The value to write to the bit: ▪ Low: `0` ▪ High: `1`

Details

Use `tsplink.writebit()` and `tsplink.writeport()` to control the output state of the trigger line when trigger operation is set to `tsplink.TRIG_BYPASS`.

If the output line is write-protected by the `tsplink.writeprotect` attribute, this command is ignored.

The reset function does not affect the present states of the TSP-Link trigger lines.

Example

```
tsplink.writebit(3, 0)
```

Sets trigger line 3 low (0).

Also see

tsplink.readbit() (on page 318)
tsplink.readport() (on page 318)
tsplink.writeport() (on page 330)
tsplink.writeprotect (on page 331)

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)`

data	Value to write to the port: 0 to 7

Details

The binary representation of `data` indicates the output pattern that is written to the I/O port. For example, a data value of 2 has a binary equivalent of 010. Line 2 is set high (1) and the other two lines are set low (0).

Write-protected lines are not changed.

Use the `tsplink.writebit()` and `tsplink.writeport()` commands to control the output state of the synchronization line when trigger operation is set to `tsplink.MODE_DIGITAL_OUT`.

The `reset()` function does not affect the present states of the trigger lines.

Example

```
tsplink.writeport(3)
```

Sets the synchronization lines 1 and 2 high (binary 011).

Also see

tsplink.readbit() (on page 318)
tsplink.readport() (on page 318)
tsplink.writebit() (on page 330)
tsplink.writeprotect (on page 331)

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	Mainframe reset	Not saved	0

Usage

```
mask = tsplink.writeprotect
tsplink.writeprotect = mask
```

mask	An integer that specifies the value of the bit pattern for write-protect; set bits to `1` to write-protect the corresponding TSP-Link trigger line

Details

The binary equivalent of *mask* indicates the mask to be set for the TSP-Link trigger line. For example, a *mask* value of `5` has a binary equivalent of 101. This *mask* write-protects TSP-Link trigger lines 1 and 3.

Example

```
tsplink.writeprotect = 5
```

Write-protects TSP-Link trigger lines 1 and 3.

Also see

Digital I/O trigger control modes (on page 79)
tsplink.readbit() (on page 318)
tsplink.readport() (on page 318)
tsplink.writebit() (on page 330)
tsplink.writeport() (on page 330)

tspnet.clear()

This function clears any pending output data from the instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.clear(connectionID)
```

connectionID	The connection ID returned from `tspnet.connect()`

Details

This function clears any pending output data from the device. No data is returned to the caller and no data is processed.

Example

```
tspnet.write(testdevice, "print([[hello]])")
print(tspnet.readavailable(testdevice))
tspnet.clear(testdevice)
print(tspnet.readavailable(testdevice))
```

Write data to a device, then print how much is available.

Output:

```
6.00000e+00
```

Clear data and print how much data is available again.

Output:

```
0.00000e+00
```

Also see

tspnet.connect() (on page 332)
tspnet.readavailable() (on page 337)
tspnet.write() (on page 342)

tspnet.connect()

This function establishes a network connection with another LAN instrument or device through the LAN interface.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
connectionID = tspnet.connect("ipAddress")
connectionID = tspnet.connect("ipAddress", portNumber, "initString")
```

connectionID	The connection ID to be used as a handle in all other `tspnet` function calls
ipAddress	A string that contains the IP address to which to connect; accepts IP address or host name when trying to connect
portNumber	Port number (default `5025`)
initString	Initialization string to send to *ipAddress*

Details

This command connects a device to another device through the LAN interface. If the *portNumber* is 23, the interface uses the telnet protocol and sets appropriate termination characters to communicate with the device.

If a *portNumber* and *initString* are provided, it is assumed that the remote device is not TSP-enabled. The MP5000 Series does not perform any extra processing, prompt handling, error handling, or sending of commands. In addition, the `tspnet.tsp.*` commands cannot be used on devices that are not TSP-enabled.

If neither a *portNumber* nor an *initString* is provided, the remote device is assumed to be a TSP-enabled device. Depending on the state of the `tspnet.tsp.abortonconnect` attribute, the MP5000 Series sends an `abort` command to the remote device on connection.

The instrument also enables TSP prompts on the remote device and event management. The instrument places remote errors and events from the TSP-enabled device in its own event queue and prefaces these events with `Remote Error`, followed by an event description.

Do not manually change either the prompt functionality (`localnode.prompts`) or show errors by changing `localnode.showerrors` on the remote TSP-enabled device. If you do this, subsequent `tspnet.tsp.*` commands using the connection may fail.

You can simultaneously connect to a maximum of 32 remote devices.

Example 1

```
instrumentID = tspnet.connect("192.0.2.1")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a TSP-enabled device.

Example 2

```
instrumentID = tspnet.connect("192.0.2.1", 1394, "*rst\r\n")
if instrumentID then
   -- Use instrumentID as needed here
   tspnet.disconnect(instrumentID)
end
```

Connect to a device that is not TSP-enabled.

Also see

localnode.prompts (on page 210)
localnode.showerrors (on page 213)
tspnet.disconnect() (on page 333)

tspnet.disconnect()

This function disconnects a specified TSP-Net session.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.disconnect(`*`connectionID`*`)`

`connectionID`	The connection ID returned from `tspnet.connect()`

Details

This function disconnects the two devices by closing the connection. The *`connectionID`* is the session handle returned by `tspnet.connect()`.

For TSP-enabled devices, this aborts any remotely running commands or scripts.

Example

```
testID = tspnet.connect("192.0.2.0")
-- Use the connection
tspnet.disconnect(testID)
```

Create a TSP-Net session. Close the session.

Also see

tspnet.connect() (on page 332)

tspnet.execute()

This function sends a command string to the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.execute("connectionID", "commandString")
value1 = tspnet.execute("connectionID", "commandString", formatString)
value1, value2 = tspnet.execute("connectionID", "commandString", formatString)
value1, ..., valueN = tspnet.execute("connectionID", "commandString", formatString)
```

connectionID	The connection ID returned from `tspnet.connect()`
commandString	The command to send to the remote device
value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The *N* th value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
formatString	Format string for the output

Details

This command sends a command string to the remote instrument. A termination is added to the command string when it is sent to the remote instrument (`tspnet.termination()`). You can also specify a format string, which causes the command to wait for a response from the remote instrument. The MP5000 Series decodes the response message according to the format specified in the format string and returns the message as return values from the function (see `tspnet.read()` for format specifiers).

When this command is sent to a TSP-enabled instrument, the MP5000 Series suspends operation until a timeout error is generated or until the instrument responds. The TSP prompt from the remote instrument is read and discarded. The mainframe places any remotely generated errors and events into its error and event queue. When the optional format string is not specified, this command is equivalent to `tspnet.write()`, except that a termination is automatically added to the end of the command.

Example 1

```
tspnet.execute(instrumentID, "runScript()")
```

Command the remote device to run a script named `runScript`.

Example 2

```
tspnet.timeout = 5
id_instr = tspnet.connect("192.0.2.23", 23, "*rst\r\n")
tspnet.termination(id_instr, tspnet.TERM_CRLF)
tspnet.execute(id_instr, "*idn?")
print("tspnet.execute returns:", tspnet.read(id_instr))
```

Print the `*idn?` string from the remote device.

Also see

tspnet.connect() (on page 332)
tspnet.read() (on page 336)
tspnet.termination() (on page 338)
tspnet.write() (on page 342)

tspnet.idn()

This function retrieves the response of the remote device to `*IDN?`.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

idnString = tspnet.idn(*connectionID*)

idnString	The returned `*IDN?` string
connectionID	The connection ID returned from `tspnet.connect()`

Details

This function retrieves the response of the remote device to `*IDN?`.

Example

```
deviceID = tspnet.connect("192.0.2.1")
print(tspnet.idn(deviceID))
tspnet.disconnect(deviceID)
```

Assume the instrument is at IP address 192.0.2.1.

The output that is produced when you connect to the instrument and read the identification string may appear as:

```
Tektronix,Model MP5103, 1398687, 1.0.0
```

Also see

tspnet.connect() (on page 332)

tspnet.read()

This function reads data from a remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value1 = tspnet.read(connectionID)
value1 = tspnet.read(connectionID, formatString)
value1, value2 = tspnet.read(connectionID, formatString)
value1, ..., valueN = tspnet.read(connectionID, formatString)
```

value1	The first value decoded from the response message
value2	The second value decoded from the response message
valueN	The *N* th value decoded from the response message; there is one return value for each format specifier in the format string
...	One or more values separated with commas
connectionID	The connection ID returned from `tspnet.connect()`
formatString	Format string for the output, maximum of 10 specifiers

Details

This command reads available data from the remote instrument and returns responses for the specified number of arguments.

The format string can contain the following specifiers:

`%[width]s`	Read data until the specified length
`%[max width]t`	Read data until the specified length or until punctuation is found, whichever comes first
`%[max width]n`	Read data until a newline or carriage return
`%d`	Read a number (delimited by punctuation)

A maximum of 10 format specifiers can be used for a maximum of 10 return values.

If *formatString* is not provided, the command returns a string that contains the data until a new line is reached. If no data is available, the MP5000 Series pauses operation until the requested data is available or until a timeout error is generated. Use `tspnet.timeout` to specify the timeout period.

When the mainframe reads from a TSP-enabled remote instrument, the mainframe removes Test Script Processor (TSP™) prompts and places any errors it receives from the remote instrument into its own event log. The mainframe prefaces errors from the remote device with "`Remote Error`," followed by the event number and description.

Example

```
tspnet.write(deviceID, "*idn?\r\n")
print("write/read returns:", tspnet.read(deviceID))
```

Send the "*idn?\r\n" message to the instrument connected as deviceID.

Display the response that is read from deviceID (based on the *idn? message).

Also see

tspnet.connect() (on page 332)
tspnet.readavailable() (on page 337)
tspnet.timeout (on page 339)
tspnet.write() (on page 342)

tspnet.readavailable()

This function checks if output data is available from the remote device.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

bytesAvailable = tspnet.readavailable(*connectionID*)

bytesAvailable	The number of bytes available to be read from the connection
connectionID	The connection ID returned from tspnet.connect()

Details

No data is read from the instrument. This allows TSP scripts to continue to run without waiting on a remote command to finish.

Example

```
ID = tspnet.connect("192.0.2.1")
tspnet.write(ID, "*idn?\r\n")
repeat bytes = tspnet.readavailable(ID) until bytes > 0
print(tspnet.read(ID))
tspnet.disconnect(ID)
```

Send commands that create data.

Wait for data to be available.

Also see

tspnet.connect() (on page 332)
tspnet.read() (on page 336)

tspnet.reset()

This function disconnects all TSP-Net sessions.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

tspnet.reset()

Details

This command disconnects all remote instruments connected through TSP-Net. For TSP-enabled devices, this causes any commands or scripts running remotely to be terminated.

Also see

None

tspnet.termination()

This function sets the device line termination sequence.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
type = tspnet.termination(connectionID)
type = tspnet.termination(connectionID, termSequence)
```

type	An enumerated value indicating the termination type: ▪ 1 or `tspnet.TERM_LF` ▪ 4 or `tspnet.TERM_CR` ▪ 2 or `tspnet.TERM_CRLF` ▪ 3 or `tspnet.TERM_LFCR`
connectionID	The connection ID returned from `tspnet.connect()`
termSequence	The termination sequence

Details

This function sets and gets the termination character sequence that is used to indicate the end of a line for a TSP-Net connection.

Using the *termSequence* parameter sets the termination sequence. The present termination sequence is always returned.

For the *termSequence* parameter, use the same values as *type*. There are four possible combinations, all of which are made up of line feeds (LF or 0x10) and carriage returns (CR or 0x13). For TSP-enabled devices, the default is `tspnet.TERM_LF`. For devices that are not TSP-enabled, the default is `tspnet.TERM_CRLF`.

Example

```
deviceID = tspnet.connect("192.0.2.1")
if deviceID then
   tspnet.termination(deviceID, tspnet.TERM_LF)
end
```

Sets termination type for IP address 192.0.2.1 to `TERM_LF`.

Also see

tspnet.connect() (on page 332)
tspnet.disconnect() (on page 333)

tspnet.timeout

This attribute sets the timeout value for the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Mainframe reset	Not saved	20.0 (20 s)

Usage

```
value = tspnet.timeout
tspnet.timeout = value
```

value	The timeout duration in seconds: 0.001 to 30.0

Details

This attribute sets the amount of time the `tspnet.connect()`, `tspnet.execute()`, and `tspnet.read()` commands wait for a response.

The time is specified in seconds. The timeout may be specified to millisecond resolution but is only accurate to the nearest 10 ms.

Example

```
tspnet.timeout = 2.0
```

Sets the timeout duration to 2 s.

Also see

tspnet.connect() (on page 332)
tspnet.execute() (on page 334)
tspnet.read() (on page 336)

tspnet.tsp.abort()

This function causes the TSP-enabled instrument to stop executing any of the commands that were sent to it.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
tspnet.tsp.abort(connectionID)
```

connectionID	Integer value used as a handle for other `tspnet` commands

Details

This function is appropriate only for TSP-enabled instruments.

Sends an abort command to the remote instrument.

Example

```
tspnet.tsp.abort(testConnection)
```

Stops remote instrument execution on `testConnection`.

Also see

None

tspnet.tsp.abortonconnect

This attribute contains the setting for abort on connect to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	No	Mainframe reset	Not saved	1 (enable)

Usage

```
tspnet.tsp.abortonconnect = value
value = tspnet.tsp.abortonconnect
```

value	▪ Enable: `1` ▪ Disable: `0`

Details

This setting determines if the instrument sends an abort message when it attempts to connect to a TSP-enabled instrument using the `tspnet.connect()` function.

When you send the abort command on an interface, it causes any other active interface on that instrument to close. If you do not send an abort command (or if `tspnet.tsp.abortonconnect` is set to 0) and another interface is active, connecting to a TSP-enabled remote instrument results in a connection. However, the instrument does not respond to subsequent reads or executes because control of the instrument is not obtained until an abort command has been sent.

Example

```
tspnet.tsp.abortonconnect = 0
```

Configure the instrument so that it does not send an abort command when connecting to a TSP-enabled instrument.

Also see

tspnet.connect() (on page 332)

tspnet.tsp.rbtablecopy()

This function copies a reading buffer synchronous table from a remote instrument to a TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
table = tspnet.tsp.rbtablecopy(connectionID, "name")
table = tspnet.tsp.rbtablecopy(connectionID, "name", startIndex, endIndex)
```

table	A copy of the synchronous table or a string
connectionID	Integer value used as a handle for other `tspnet` commands
name	The full name of the reading buffer name and synchronous table to copy
startIndex	Integer start value
endIndex	Integer end value

Details

This function is only appropriate for TSP-enabled instruments.

This function reads the data from a reading buffer on a remote instrument and returns an array of numbers or a string representing the data. The *startIndex* and *endIndex* parameters specify the portion of the reading buffer to read. If no index is specified, the entire buffer is copied.

The function returns a table if the table is an array of numbers; otherwise, a comma-delimited string is returned.

This command is limited to transferring 50,000 readings at a time.

Example

```
t = tspnet.tsp.rbtablecopy(testConnection, "testRemotebuffername.readings", 1, 3)
print(t[1], t[2], t[3])
```

Copy the specified readings table for buffer items 1 through 3, then display the first three readings.

Example output:

```
4.56534e-01
4.52675e-01
4.57535e-01
```

Also see

None

tspnet.tsp.runscript()

This function loads and runs a script on a remote TSP-enabled instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
tspnet.tsp.runscript(connectionID, "script")
tspnet.tsp.runscript(connectionID, "name", "script")
```

connectionID	Integer value used as an identifier for other `tspnet` commands
name	The name that is assigned to the script
script	The body of the script as a string

Details

This function is appropriate only for TSP-enabled instruments.

This function downloads a script to a remote instrument and runs it. It automatically adds the appropriate `loadscript` and `endscript` commands around the script, captures any errors, and reads back any prompts. No additional substitutions are done on the text.

The script is automatically loaded, compiled, and run.

Any output from previous commands is discarded.

This command does not wait for the script to complete.

If you do not want the script to do anything immediately, make sure the script only defines functions for later use. Use the `tspnet.execute()` function to execute those functions later.

If the remote instrument supports anonymous scripts, if no name is specified, the script is loaded as the anonymous script.

Example

```
tspnet.tsp.runscript(myconnection, "mytest",
"print([[start]]) for d = 1, 10 do print([[work]]) end print([[end]])")
```

Load and run a script entitled `mytest` on the TSP-enabled instrument connected with `myconnection`.

Also see

tspnet.execute() (on page 334)

tspnet.write()

This function writes a string to the remote instrument.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`tspnet.write(`*`connectionID`*`, "`*`inputString`*`")`

`connectionID`	The connection ID returned from `tspnet.connect()`
`inputString`	The string to be written

Details

The `tspnet.write()` function sends *`inputString`* to the remote instrument. It does not wait for command completion on the remote instrument.

The MP5000 Series sends *`inputString`* to the remote instrument exactly as indicated. The *`inputString`* must contain any necessary new lines, termination, or other syntax elements needed to complete properly.

Because `tspnet.write()` does not process output from the remote instrument, do not send commands that generate too much output without processing the output. This command can stop executing if there is too much unprocessed output from previous commands.

Example

```
tspnet.write(myID, "runscript()\r\n")
```

Commands the remote instrument to execute a command or script named `runscript()` on a remote device identified in the system as `myID`.

Also see

tspnet.connect() (on page 332)
tspnet.read() (on page 336)

usbtmc.access

This command enables or disables access to the mainframe rear-panel USB port.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	lan.reset() lan.restoredefaults()	Nonvolatile memory	localnode.ENABLE

Usage

```
state = usbtmc.access
usbtmc.access = state
```

state	Rear-panel USB port communications state: ▪ Enable USB communications: `localnode.ENABLE` ▪ Enable USB communications with password: `localnode.PROTECT`

Details

When set to `localnode.ENABLE`, this command enables communications through the USB-C port on the rear panel of the instrument without logging in. When this command is set to `localnode.PROTECT`, you must log in with the set password before communicating through the USB-C port.

Example

```
usbtmc.access = localnode.ENABLE
```

Enables communications through the USB-C port on the mainframe.

Also see

lan.enable (on page 193)
lan.hislip.access (on page 193)
lan.rawsocket.access (on page 197)
lan.telnet.access (on page 205)

user.password.change()

This function is used to change the password that is used with the `login` command.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
user.password.change("newpassword")
```

newpassword	A string that contains the present password

Details

A password is required for any communication interface or protocol that has its access set to `localnode.PROTECT`.

A password is always required to change settings on the web interface for the MP5000 Series mainframe. The password is reset to factory default by `lan.reset()` or by pressing the LAN reset button on the front panel of the MP5103. If you reset the password to the factory default, you must set a password upon the next use.

A password must be from 1 to 20 characters. All alphanumeric characters are valid. You cannot set the password to `admin`.

The special characters `! @ # $ % ^ & *` are also valid.

Example

```
user.password.change("N3wpa55w0rd")
```

Changes the login password to `N3wpa55w0rd`.

Also see

lan.reset() (on page 198)
login (on page 213)
logout (on page 214)

userstring.add()

This function adds a user-defined string to nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`userstring.add("name", "value")`

name	The name of the string; the key of the key-value pair
value	The string to associate with *name*; the value of the key-value pair

Details

This function associates the string *value* with the string *name* and stores this key-value pair in nonvolatile memory.

Use the `userstring.get()` function to retrieve the *value* associated with the specified *name*.

You can use the `userstring` functions to store custom, instrument-specific information in the instrument, such as department number, asset number, or manufacturing plant location.

Example

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name in userstring.table() do
   print(name .. " = " ..
      userstring.get(name))
end
```

Stores user-defined strings in nonvolatile memory and recalls them from the instrument using a for loop.

Example output:

```
assetnumber = 236
contact = John Doe
product = Widgets
```

Also see

userstring.delete() (on page 345)
userstring.get() (on page 345)
userstring.table() (on page 346)

userstring.delete()

This function deletes a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstring.delete("name")
```

name	The name (key) of the key-value pair of the user-defined string to delete

Details

This function deletes the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.delete("assetnumber")
userstring.delete("product")
userstring.delete("contact")
```

Deletes the user-defined strings associated with the `assetnumber`, `product`, and `contact` names.

Also see

userstring.add() (on page 344)
userstring.get() (on page 345)
userstring.table() (on page 346)

userstring.get()

This function retrieves a user-defined string from nonvolatile memory.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
value = userstring.get("name")
```

value	The value of the user-defined string key-value pair
name	The name (key) of the user-defined string

Details

This function retrieves the string that is associated with *name* from nonvolatile memory.

Example

```
userstring.add("assetnumber", "236")
value = userstring.get("assetnumber")
print(value)
```

Create the user-defined string `assetnumber`, set to a value of `236`.

Read the value associated with the user-defined string named `assetnumber`.

Store it in a variable called `value`, then print the variable `value`.

Output:

```
236
```

Also see

userstring.add() (on page 344)
userstring.delete() (on page 345)
userstring.table() (on page 346)

userstring.table()

This function creates a table for the user-defined string catalog.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
userstringtable = userstring.table()
for name, value in pairs(userstring.table()) do body end
```

userstringtable	A table that holds copies of all user string pairs
name	The name of the string; the key of the key-value pair
body	Code to execute in the body of the for loop

Details

The table provides access for user-defined string pairs, allowing you to manipulate all the key-value pairs in nonvolatile memory. The entries are enumerated in no particular order.

Example 1

```
for name, value in pairs(userstring.table()) do
   userstring.delete(name)
end
```

Deletes all user-defined strings in nonvolatile memory.

Example 2

```
userstring.add("assetnumber", "236")
userstring.add("product", "Widgets")
userstring.add("contact", "John Doe")
for name, value in pairs(userstring.table()) do
   print(name .. " = " ..
      "value")
end
```

Prints all `userstring` key-value pairs.

Output:

```
product = Widgets
assetnumber = 236
contact = John Doe
```

Notice the key-value pairs are not listed in the order they were added.

Also see

userstring.add() (on page 344)
userstring.delete() (on page 345)
userstring.get() (on page 345)

waitcomplete()

This function waits for all previously started overlapped commands to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
waitcomplete()
waitcomplete(group)
```

group	Specifies the TSP-Link group on which to wait

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `waitcomplete()` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. This command is not needed for sequential commands.

A group number may only be specified when this node is the master node.

If no *group* is specified, the local group is used.

If zero (0) is specified for the *group*, this function waits for all nodes in the system.

Example

```
waitcomplete()
```

Waits for all nodes in the local group.

Also see

slot[Z].psu[X].measure.overlappedY() (on page 384)
slot[Z].psu[X].overlapped (on page 390)
slot[Z].smu[X].measure.overlappedY() (on page 500)
slot[Z].smu[X].overlapped (on page 505)

PSU module TSP commands

Introduction

he TSP commands available for the MP5000 Series power supply unit (PSU) modules are listed in alphabetical order.

bufferVar.appendmode

This attribute sets the state of the append mode of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0 (off)

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

state	The reading buffer append mode; set to one of the following: ▪ Append mode off; clear the buffer and then add new data to buffer: `0` ▪ Append mode on; new data is added to the existing buffer content: `1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

Assign a value to this attribute to enable or disable the buffer append mode. This value can only be changed when the buffer is empty. Use `bufferVar.clear()` to empty the buffer.

If the append mode is set to `0` (off), any previously stored readings in the buffer are cleared before new ones are stored. If append mode is set to `1`, any previously stored readings remain in the buffer and new readings are added to the buffer following the stored readings.

With append mode set to `1` (on), the first new measurement is stored at reading buffer location `[n+1]`, where *n* is the number of readings stored in the buffer. When the fill mode is set to once, measurements are only appended to a buffer if the measure count is less than the available space in the buffer. If the buffer cannot hold the readings, an error is returned and no measurements are made.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
buffer1.appendmode = 1
print(buffer1.appendmode)
```

Append new readings to contents of the reading buffer `buffer1` created for channel 1 of the module installed in slot 1.

Example output:

```
1.00000e+00
```

Also see

bufferVar.clear() (on page 351)

bufferVar.basetimestampfractional

This attribute contains the fractional seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

fractionalSec = *bufferVar*.basetimestampfractional

fractionalSec	The fractional seconds portion of the timestamp when the first reading was stored
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The read-only base timestamp attributes contain the timestamp of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 AM January 1, 1970 (UTC) that the measurement was performed and stored.

The base timestamp of the reading buffer is divided into two parts. This attribute contains the fractional portion of the base timestamp (with 100 ns of precision) and can be used with the seconds portion of the base timestamp.

Example

```
buffer = slot[1].psu[1].defbuffer1
print(tostring(buffer.basetimestampseconds))
print(tostring(buffer.basetimestampfractional))
```

Return the timestamp for the first reading stored in defbuffer1 on channel 1 of the PSU in slot 1.

Example output:

```
1756207445
0.1366606
```

Also see

bufferVar.basetimestampseconds (on page 349)
Reading buffers (on page 96)

bufferVar.basetimestampseconds

This attribute contains the whole seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

nonFracSeconds = *bufferVar*.basetimestampseconds

nonFracSeconds	The nonfractional seconds portion of the timestamp when the first reading was stored
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The read-only base timestamp attributes contain the timestamp of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 AM January 1, 1970 (UTC) that the measurement was performed and stored.

The base timestamp of the reading buffer is divided into two parts. This attribute contains the whole seconds portion of the base timestamp (resolution of 1 second.) It can be used with the fractional seconds portion of the base timestamp for additional precision.

Example

```
buffer = slot[3].psu[2].defbuffer2
print(tostring(buffer.basetimestampseconds))
```

Return the timestamp for the first reading stored in defbuffer2 on channel 2 of the PSU in slot 3.

Example output:

```
1756222247
```

This output indicates that the timestamp is 1,756,222,247 seconds or Tuesday, August 26, 2025, at 11:30:47 AM.

Also see

bufferVar.basetimestampfractional (on page 349)

bufferVar.cachemode

This attribute enables or disables the cache of the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	1 (enabled)

Usage

```
cacheMode = bufferVar.cachemode
bufferVar.cachemode = cacheMode
```

cacheMode	The reading buffer cache mode; set to one of the following: ▪ Cache mode disabled (off): `0` ▪ Cache mode enabled (on): `1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The reading buffer cache improves access speed to reading buffer data. However, if you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you initiate successive sweeps but do not reconfigure the sweep measurements. It can also happen when the *bufferVar*`.fillmode` attribute is set to `1`, which may cause reading buffer data to be overwritten.

To prevent the return of stale cache data, disable the cache or make an explicit call to *bufferVar*`.clearcache()`.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.cachemode = 0
```

Disables the reading buffer cache of user-defined reading buffer `buffer1`.

Example 2

```
slot[2].psu[2].defbuffer1.cachemode = 0
print(slot[2].psu[2].defbuffer1.cachemode)
```

Disables the reading buffer cache of default reading buffer `defbuffer1` on PSU channel 2 in slot 2.

Also see

bufferVar.clearcache() (on page 352)
bufferVar.fillmode (on page 353)

bufferVar.capacity

This attribute contains the capacity of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

maxNumber = *bufferVar*.capacity

maxNumber	The maximum number of readings the buffer can store
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This read-only attribute returns the number of readings that can be stored in the buffer. The buffer capacity does not change as readings fill the buffer.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
maxNumber = buffer1.capacity
print(maxNumber)
```

Returns the maximum number of readings that can be stored in `buffer1`:

```
1.00000e+02
```

Also see

None

bufferVar.clear()

This function empties the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar.clear()

bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This function clears all readings and associated columns that are enabled from the specified buffer. It also clears the buffer cache.

Example

```
buffer1 = slot[1].psu[1].debuffer1
buffer1.clear()
```

Clears readings from the `debuffer1` buffer on channel 1 of the module in slot 1.

Also see

bufferVar.clearcache() (on page 352)

bufferVar.clearcache()

This function clears the buffer cache of the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar`.clearcache()`

bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you:

- Initiate successive sweeps without reconfiguring the sweep measurements. Watch for this when running Lua code remotely on more than one node, because values in the reading buffer cache may change while the Lua code is running.
- Overwrite data in the reading buffer by setting the *bufferVar*`.fillmode` attribute to `1`.

To avoid this, you can include explicit calls to the *bufferVar*`.clearcache()` function to remove stale values from the reading buffer cache.

Example

```
buffer1 = slot[1].psu[1].defbuffer1
buffer1.clearcache()
```

Clears all readings from the reading buffer cache of `defbuffer1` on channel 1 of the module in slot 1.

Also see

bufferVar.fillmode (on page 353)

bufferVar.fillcount

This attribute sets the reading buffer fill mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
fillCount = bufferVar.fillcount
bufferVar.fillcount = fillCount
```

fillCount	The reading buffer fill count; set to `0` to use the full capacity of the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is `0`, the full capacity of the buffer is used. Use this attribute to control when the PSU restarts filling the buffer at index 1, rather than having it restart when the buffer becomes full.

If the *bufferVar*`.fillcount` attribute is set to a value higher than the capacity of the buffer, the behavior is the same as having a fill count of 0. When the buffer capacity is reached, storage continues at the starting index.

This attribute is only used if *bufferVar*`.fillmode` is set to 1.

Example

```
buffer1 = slot[1].psu[1].defbuffer1
buffer1.fillmode = 1
buffer1.fillcount = 50
```

Sets the fill count for `defbuffer1` on channel 1 of the module installed in slot 1 to 50.

Also see

bufferVar.fillmode (on page 353)

bufferVar.fillmode

This attribute determines if old data is overwritten when the reading buffer is filled.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

fillMode	The reading buffer fill mode, set to one of the following: ▪ Fill once (do not overwrite old data): 0 ▪ Fill continuously; new readings restart at index 1 after acquiring reading at index `bufferVar.fillcount: 1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

When this attribute is set to fill once (`0`), the reading buffer does not overwrite readings. If the requested measurement operation would exceed the size of the buffer, no measurements are made and an error is returned.

When this attribute is set to fill continuously (`1`) and append mode is set on, new readings are added sequentially until the buffer holds *`bufferVar`*`.fillcount` elements. The next reading overwrites the reading at index 1, the following reading overwrites the reading at index 2, and so on.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.fillmode = 1
```

Enable window fill mode for `buffer1`. The readings will fill the buffer until the fillcount index and then wrap around.

Also see

bufferVar.appendmode (on page 348)
bufferVar.fillcount (on page 353)

bufferVar.measurefunctions

This attribute contains the measurement function that was used to acquire readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

`measurefunction` `=` *`bufferVar`*`.measurefunctions[`*`N`*`]`

`measurefunction`	The measurement function used (current, voltage, ohms, or watts) to acquire reading number *`N`* in the specified buffer
`bufferVar`	The reading buffer; can be a user-defined or default reading buffer
`N`	The reading number: `1` to *`bufferVar`*`.n`

Details

The `measurefunctions` buffer column is a Lua table array of strings that indicates the function that was measured for the reading.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.measurefunctions)
```

Prints the measure functions for the first five readings in a reading buffer for channel 1 of the module installed in slot 1. Example output is:

```
Voltage, Voltage, Voltage, Voltage, Voltage
```

Also see

None

bufferVar.measureranges

This attribute contains the measurement range values that were used for readings stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle bufferVar.clear()	Not applicable	Not applicable

Usage

measurerange = *bufferVar*.measureranges[*N*]

measurerange	The measurement range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to bufferVar.n

Details

The measureranges buffer column is a Lua table array of full-scale range values for the measure range that was used when the measurement was made.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.measureranges)
```

Prints the measure range for the first five readings in a reading buffer for channel 1 of the module installed in slot 1. Example output:

```
5.00000e+01, 5.00000e+01, 5.00000e+01, 5.00000e+01, 5.00000e+01
```

Also see

None

bufferVar.n

This attribute contains the number of readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

numberOfReadings = *bufferVar*.n

numberOfReadings	The number of readings stored in the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This read-only attribute contains the number of readings stored in the buffer.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, buffer1.n, buffer1.readings)
```

Create reading buffer `buffer1`.

Clear the buffer.

Set the measure count to 20.

Make voltage measurements and store them in `buffer1`.

Return all readings in the reading buffer. Example output:

```
-1.06509e-03, -1.01642e-03, -1.05698e-03, -9.02847e-04, -1.00831e-03, -9.83969e-04
, -9.92081e-04, -8.94735e-04, -9.75856e-04, -8.86622e-04, -1.05698e-03, -9.51520e-
04, -1.08131e-03, -1.05698e-03, -9.92081e-04, -1.00831e-03, -9.19071e-04, -9.19071
e-04, -1.07320e-03, -9.59632e-04
```

Also see

None

bufferVar.readings

This attribute contains the acquired readings stored in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

reading = *bufferVar*.readings[*N*]

reading	The value of the reading in the specified reading buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: `1` to *bufferVar*.n

Details

The readings buffer column is a Lua table array of the readings stored in the reading buffer. This column holds the same data that is returned when the reading buffer is accessed directly; that is, *bufferVar*[2] and *bufferVar*.readings[2] access the same value.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, buffer1.n, buffer1.readings)
```

Create reading buffer `buffer1`.

Clear the buffer.

Set the measure count to 20.

Make voltage measurements and store them in `buffer1`.

Return all readings in the reading buffer. Example output:

```
-1.06509e-03, -1.01642e-03, -1.05698e-03, -9.02847e-04, -1.00831e-03, -9.83969e-04
, -9.92081e-04, -8.94735e-04, -9.75856e-04, -8.86622e-04, -1.05698e-03, -9.51520e-
04, -1.08131e-03, -1.05698e-03, -9.92081e-04, -1.00831e-03, -9.19071e-04, -9.19071
e-04, -1.07320e-03, -9.59632e-04
```

Also see

bufferVar.n (on page 355)

bufferVar.sourcefunctions

This attribute contains the source function that was used when the readings were stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcefunction = *bufferVar*.sourcefunctions[*N*]

sourcefunction	The source function used (`Voltage`)
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: `1` to *bufferVar*.n

Details

The `sourcefunctions` buffer column is a Lua table array of strings that indicate the source function at the time of the measurement.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(10)
slot[1].psu[1].measure.count = 10
slot[1].psu[1].source.levelv = 2.2
slot[1].psu[1].source.output = slot[1].psu[1].ON
slot[1].psu[1].measure.v(buffer1)
slot[1].psu[1].source.output = slot[1].psu[1].OFF
printbuffer(1, buffer1.n, buffer1.readings, buffer1.sourcefunctions, buffer1.sourc
eoutputstates)
```

Create the reading buffer `buffer1` on slot 1 PSU channel 1.

Set the measure count to 20.

Set the source voltage level to 2.2.

Turn the output on.

Make measurements and store them in `buffer1`.

Turn the output off.

Show each reading with the source function and source output state.

Example output:

```
2.19807e+00, Voltage, On, 2.19804e+00, Voltage, On, 2.19812e+00, Voltage, On, 2.19
819e+00, Voltage, On, 2.19811e+00, Voltage, On, 2.19811e+00, Voltage, On, 2.19813e
+00, Voltage, On, 2.19817e+00, Voltage, On, 2.19816e+00, Voltage, On, 2.19817e+00,
 Voltage, On
```

Example 2

```
printbuffer(1, 10, slot[1].psu[1].defbuffer1.sourcefunctions)
```

Example output:

```
Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, Voltage, V
oltage
```

Also see

None

bufferVar.sourceoutputstates

This attribute indicates the state of the source output for readings stored in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

state = *bufferVar*.sourceoutputstates[*N*]

state	The output state when reading *N* of the specified buffer was acquired: `On` or `Off`
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: `1` to *bufferVar*.n

Details

The `sourceoutputstates` buffer column is a Lua table array of strings that indicate the state of the source output (off or on) at the time of the measurement.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(10)
slot[1].psu[1].measure.count = 10
slot[1].psu[1].source.levelv = 2.2
slot[1].psu[1].source.output = slot[1].psu[1].ON
slot[1].psu[1].measure.v(buffer1)
slot[1].psu[1].source.output = slot[1].psu[1].OFF
printbuffer(1, buffer1.n, buffer1.readings, buffer1.sourcefunctions, buffer1.sourceoutputstates)
```

Create the reading buffer `buffer1` on slot 1 PSU channel 1.

Set the measure count to 20.

Set the source voltage level to 2.2.

Turn the output on.

Make measurements and store them in `buffer1`.

Turn the output off.

Show each reading with the source function and source output state.

Example output:

```
2.19807e+00, Voltage, On, 2.19804e+00, Voltage, On, 2.19812e+00, Voltage, On, 2.19819e+00, Voltage, On, 2.19811e+00, Voltage, On, 2.19811e+00, Voltage, On, 2.19813e+00, Voltage, On, 2.19817e+00, Voltage, On, 2.19816e+00, Voltage, On, 2.19817e+00, Voltage, On
```

Also see

bufferVar.n (on page 355)

bufferVar.sourceranges

This attribute contains the source range used for readings stored in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcerange = *bufferVar*.sourceranges[*N*]

sourcerange	The source range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The sourceranges buffer recall attribute is a Lua table array of full-scale range values for the source range used when each measurement was made.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(20)
buffer1.clear()
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.readings, buffer1.sourceranges)
```

Create reading buffer buffer1 that holds 20 readings.

Clear the buffer.

Set the measure count to 20.

Make voltage measurements and store them in `buffer1`.

Print the readings and source ranges.

Example output:

```
-9.19071e-04, 5.00000e+01, -9.75856e-04, 5.00000e+01, -8.94735e-04, 5.00000e+01, -
9.83969e-04, 5.00000e+01, -9.27183e-04, 5.00000e+01
```

Also see

bufferVar.n (on page 355)

bufferVar.sourcevalues

This attribute contains the source levels that were programmed when the readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcevalue = *bufferVar*.sourcevalues[*N*]

sourcevalue	The source value programmed while acquiring reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The `sourcevalues` buffer column is a Lua table of the programmed sourced value at the time of the reading.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].psu[1].makebuffer(20)
slot[1].psu[1].source.levelv = 3.5
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.sourcevalues)
```

Create buffer1 that contains 20 readings.

Set the voltage source level to 3.5.

Set the measurement count to 20

Make voltage measurements and store them in buffer1.

Return the first 5 source values in buffer1:

```
3.50000e+00, 3.50000e+00, 3.50000e+00, 3.50000e+00, 3.50000e+00
```

Also see

bufferVar.n (on page 355)

bufferVar.statuses

This attribute contains the status values of readings in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

statusInformation = *bufferVar*.statuses[*N*]

statusInformation	The status value when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar.n*

Details

This buffer column is a Lua table of the status values for each reading in the buffer. The status values are floating-point numbers that encode the status value.

The buffer status values are not updated during trigger model operation.

The buffer status bits are listed in the following table.

Bit	Name	Decimal value	Hex value	Description
B0	Reserved	1	0x01	Reserved for future use
B1	Overtemp	2	0x02	Overtemperature shutdown; measurement made with output off
B2	Reserved	4	0x04	Reserved for future use
B3	Reserved	8	0x08	Reserved for future use
B4	Reserved	16	0x10	Reserved for future use
B5	Rel	32	0x20	Relative offset applied to reading
B6	Limit	64	0x40	Current limit in effect; output voltage was reduced
B7	Reserved	128	0x80	Reserved for future use

Example

```
buffer1 = slot[1].psu[1].makebuffer(20)
slot[1].psu[1].source.levelv = 3.5
slot[1].psu[1].measure.count = 20
slot[1].psu[1].measure.rel.enablev = slot[1].psu[1].ENABLE
slot[1].psu[1].measure.rel.levelv = slot[1].psu[1].measure.v()
slot[1].psu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.readings)
printbuffer(1, 5, buffer1.statuses)
```

Create a buffer. Set the source level to 3.5 V.

Set the measure count to 20.

Enable the relative offset and set the relative offset to the voltage measurement.

Measure voltage.

Return the readings and buffer status values. Example output:

```
-6.48976e-05, -1.62245e-05, -8.11229e-06, -9.37774e-11, -9.73464e-05
3.20000e+01, 3.20000e+01, 3.20000e+01, 3.20000e+01, 3.20000e+01
```

Also see

slot[Z].psu[X].measure.rel.enableY (on page 386)
slot[Z].psu[X].source.limiti (on page 393)

bufferVar.timestampresolution

This attribute contains the resolution of the timestamp for each reading stored in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	100e-9 (100 ns)

Usage

```
resolution = bufferVar.timestampresolution
bufferVar.timestampresolution = resolution
```

resolution	Timestamp resolution in seconds (minimum 100 ns; rounded to an even power of 200 ns)
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

Assigning a value to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value.

This value can only be changed when the buffer is empty. Empty the buffer using the *bufferVar.clear()* function.

The finest timestamp resolution is 100e−9 seconds (100 ns). At this resolution, the reading buffer can store unique timestamps for up to 7 minutes. You can increase this value for long tests.

The value specified when setting this attribute is rounded to an even power of 200 ns.

This setting does not affect the resolution of the base timestamp of the reading buffer.

Example

```
buffer1 = slot[1].psu[1].makebuffer(100)
buffer1.clear()
buffer1.timestampresolution = 8e-6
print(buffer1.timestampresolution)
```

Creates a new reading buffer and sets its timestamp resolution to 8 µs. Due to rounding, the actual resolution is set to 12.8 µs.

Example output:

```
1.28000e-05
```

Also see

bufferVar.clear() (on page 351)
bufferVar.timestamps (on page 363)

bufferVar.timestamps

This attribute contains the relative timestamp for each reading in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

timestamp = *bufferVar*.timestamps[*N*]

timestamp	The timestamp of reading number *N* in the specified reading buffer when the reading was acquired, relative to the first reading in the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

This buffer recall attribute is a Lua table that contains the timestamp of when each reading occurred. The timestamp is in seconds relative to the first measurement added to the reading buffer. It is collected at the beginning of the aperture time of each measurement.

The resolution of the timestamp is set by the *bufferVar*.timestampresolution attribute.

When used with the base timestamp attributes, you can calculate the absolute timestamp of each measurement.

Example

```
buffer = slot[1].psu[1].defbuffer2
print(buffer.timestamps[2])
printbuffer(1, buffer.n, buffer.timestamps)
```

Retrieve the timestamp of the second reading stored in defbuffer2 on channel 1 of the PSU in slot 1.

Retrieve the timestamps of each reading stored in the same buffer.

Example output:

```
6.63871e-02
0.00000e+00, 6.63871e-02, 1.32775e-01, 1.99160e-01, 2.65548e-01, 3.31938e-01, 3.98327e-01, 4.64711e-01, 5.31093e-01, 5.97480e-01
```

This output indicates that the second measurement occurred 66.3871 ms after the first measurement, then lists the timestamp for each measurement in slot[1].psu[1].defbuffer2 relative to its first measurement.

Also see

bufferVar.basetimestampfractional (on page 349)
bufferVar.basetimestampseconds (on page 349)
bufferVar.clear() (on page 351)
bufferVar.n (on page 355)
bufferVar.readings (on page 356)
bufferVar.statuses (on page 361)
bufferVar.timestampresolution (on page 362)
Reading buffers (on page 96)

slot[Z].bank[X].meminfo()

This function returns memory information for a specified module bank.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

memory = slot[*Z*].bank[*X*].meminfo()

memory	Contains information about the memory usage for the specified memory bank
Z	Module slot number
X	Module memory bank number

Details

Use this command to retrieve a table of the allocated or used memory for a module bank.

The returned table contains the following fields:

- `total`: The total memory available for allocation
- `readingbuffer`: Memory in use by reading buffers
- `used`: The total memory in use
- `configlist`: Memory in use by configuration lists

PSU modules use one bank for both channels (bank `1`).

Example 1

```
memory = slot[1].bank[1].meminfo()
for i, value in pairs(memory) do
print(i, value)end
```

Queries and retrieves all lists of memory values on bank 1 of the module installed in slot 1.

Output:

```
total             1.02400e+05
readingbuffer     4.35200e+03
used              4.35200e+03
configlist        0.00000e+00
```

Example 2

```
memory = slot[1].bank[1].meminfo()
print(memory.used)
```

Retrieves the total amount of memory used for bank 1 of the module installed in slot 1.

Output:

```
used              4.35200e+03
configlist        0.00000e+00
```

Also see

Memory considerations for the runtime environment (on page 42)

slot[Z].firmware.update()

This function updates the firmware of a module after the firmware file is copied to the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].firmware.update()`

Z	Module slot number

Details

This command updates the module firmware. If the update is successful, the module is also restarted.

Before issuing this command, you need to load the firmware file onto the instrument. To download an image, use the mainframe `flash` and `endflash` keyword sequence or the `firmware.image.load()` command.

To update multiple modules with the same firmware file, you can send this command multiple times with different slot numbers.

For more information, see the applicable Reference Manual for your module, available from tek.com.

CAUTION: Do not turn off power until the update process is complete.

Example

```
firmware.image.load("/usb1/module/MPSU50-2ST-1.0.0.x")
slot[1].firmware.update()
```

Load the module firmware `MPSU50-2ST-1.0.0.x` file in the `module` folder of the USB drive inserted into the front of the mainframe.

Update the module in slot 1.

Also see

firmware.image.load() (on page 167)
slot[Z].firmware.verify (on page 366)

slot[Z].firmware.verify()

This function verifies that the firmware update succeeded.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

result = slot[*Z*].firmware.verify()

result	Status of the firmware verification: ▪ Firmware not correctly installed: false ▪ Firmware update succeeded: true
Z	Module slot number

Details

This function verifies that the firmware has updated correctly after you run the slot[*Z*].firmware.update() command.

This function compares the firmware downloaded to the mainframe and firmware in the module in the specified slot. If they are equal, the function returns true. Otherwise, it returns false.

You can use this command to determine that the firmware has flashed correctly after performing the slot[*Z*].firmware.update() command without downloading the image again.

Example

```
print(slot[1].firmware.verify())
```

Verifies that the firmware of the module installed in slot 1 updated correctly.

Also see

slot[Z].firmware.update (on page 365)

slot[Z].license

This attribute stores the license texts for the module installed in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

license = slot[*Z*].license

license	License text for the instrument
Z	Module slot number

Details

This attribute contains a large amount of text, including the Tektronix End User License Agreement and any open source software licenses in effect.

Example

```
print(slot[1].license)
```

Returns the text of the licenses for the module installed in slot 1 of the mainframe.

Also see

None

slot[Z].manufacturer

This attribute stores the manufacturer of the module in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Tektronix

Usage

manufacturer = slot[*Z*].manufacturer

manufacturer	This is always TEKTRONIX
Z	Module slot number

Example

```
print(slot[1].manufacturer)
```

Returns the manufacturer of the module installed in slot 1:

```
TEKTRONIX
```

Also see

None

slot[Z].model

This attribute stores the model number of the module in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

model = slot[*Z*].model

model	The model number of the module
Z	Module slot number

Example

```
print(slot[1].model)
```

Returns the model number of the module installed in slot 1.

Also see

None

slot[Z].psu[X].abort()

This function terminates all overlapped operations on the specified channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].abort()`

Z	Module slot number
X	Module channel number

Details

This function terminates all overlapped operations on the specified channel.

This function does not turn the output off or change any settings.

Example

```
slot[1].psu[1].overlapped = slot[1].psu[1].ENABLE
slot[1].psu[1].source.levelv = 5
slot[1].psu[1].source.slewrate.v = 1
slot[1].psu[1].source.output = slot[1].psu[1].ON
delay(1)
slot[1].psu[1].abort()
print(slot[1].psu[1].source.levelv) -- This will not reach 5 V.

slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

Enable overlapped mode for channel 1 of the module installed in slot 1.

Set the source level to 5 V and the slew rate to 1 V/s.

Turn the source output on.

Set a 1 second delay.

Abort the operations.

Print the source level value.

Turn the output off.

Also see

None

slot[Z].psu[X].config.active()

This function returns a list of the active channel settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

config = slot[*Z*].psu[*X*].config.active()

config	A table of attributes representing the active configuration of the channels
Z	Module slot number
X	Module channel number

Details

Use this command to return a configuration of all active attributes on the selected channel. The configuration is represented by a Lua table. This configuration can be used to recall stored settings to a channel, added to an existing configuration list, or used as a reference when creating a new configuration list.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].psu[1].configlist.create("MyConfigList")
ch1Config = slot[1].psu[1].config.active()
slot[1].psu[1].configlist.store("MyConfigList", 1, ch1Config)
```

Creates a configuration list named MyConfigList.

Saves the attribute set on channel 1 of slot 1 to `ch1Config`.

Stores that configuration to the configuration list `myConfigList` in index 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].config.default() (on page 369)
slot[Z].psu[X].config.set() (on page 370)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.store() (on page 378)

slot[Z].psu[X].config.default()

This function recalls a configuration that represents the default channel settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

config = slot[*Z*].psu[*X*].config.default()

config	A set of attributes representing the default configuration

Z	Module slot number
X	Module channel number

Details

Use this command to return a table of the default configuration settings for a channel. The configuration is represented by a Lua table. This configuration can be used to recall stored settings to a channel, added to an existing configuration list or used as a reference when creating a new configuration list.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
startConfig = slot[1].psu[1].config.default()
```

Saves the default configuration for channel 1 of the module installed in slot 1 to the variable `startConfig`.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].config.active() (on page 369)
slot[Z].psu[X].config.set() (on page 370)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.store() (on page 378)

slot[Z].psu[X].config.set()

This function sets a channel configuration based on a table of attributes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].config.set(config)`

Z	Module slot number
X	Module channel number
config	A table of attributes

Details

This function sets the active channel configuration based on a defined table. If the table is missing entries, those entries are not changed.

Example

```
slot[1].psu[1].config.set(setup1)
```

Configures channel 1 of the module installed in slot 1 to the settings in the configuration table `setup1`.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].config.active() (on page 369)
slot[Z].psu[X].config.default() (on page 369)

slot[Z].psu[X].configlist.append()

This function adds a step to the end of a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].psu[X].configlist.append("configListName")
slot[Z].psu[X].configlist.append("configListName", config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
config	A table of attributes to be added to the configuration list.

Details

Use this function to add a step to the end of a configuration list. If *config* is not specified, the active configuration is used.

Any attributes not specified in *config* will inherit the values specified by the reference configuration used when the configuration list was created.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].psu[1].configlist.append("MyConfigList","measureConfig2")
```

Adds the table of attributes in `measureConfig2` to the end of the configuration list `MyConfigList` on channel 1 of the module in slot 1.

Also see

slot[Z].psu[X].config.active() (on page 369)
slot[Z].psu[X].config.default() (on page 369)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.deletelist() (on page 374)

slot[Z].psu[X].configlist.create()

This function creates a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].psu[X].configlist.create("configListName")
slot[Z].psu[X].configlist.create("configListName", config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list; must be unique on this channel
config	A table of attributes to be used as the reference configuration

Details

This command creates a new configuration list. If *config* is not specified, the active channel configuration will be used as the reference configuration. The reference configuration is used to fill in unspecified settings when additional configurations are added. To add configuration indexes to this list, use the `slot[Z].psu[X].configlist.store()` or `slot[Z].psu[X].configlist.append()` commands.

The *configListName* parameter must be unique on the specified slot and channel. If *configListName* specifies an existing configuration list, an error is generated. The original configuration list is not overwritten.

Configuration lists are not saved when the instrument is turned off. They are preserved through resets.

Example

```
slot[1].psu[1].configlist.create("MyConfigList", "drainConfig")
```

Creates a configuration list named `MyConfigList` based on the configuration `drainConfig` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].config.active() (on page 369)
slot[Z].psu[X].config.default() (on page 369)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.store() (on page 378)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.delete()

This function deletes a step in a configuration list at the specified index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].configlist.delete("configListName", index)`

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list

Details

Deletes an index from a configuration list. An index must be specified.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the highest numbered index first, then the next lowest index, and so on.

When deleting several consecutive indexes, it is best to delete the lowest number repeatedly until the indexes are removed.

To delete a configuration list, use the `slot[Z].psu[X].configlist.deletelist()` command.

The former functionality of this command is supported but deprecated beginning with mainframe firmware version 1.1.0. The former functionality allowed you to delete an entire configuration list by omitting the *index* parameter.

Example

```
slot[1].psu[1].configlist.delete("myConfigList", 2)
```

Deletes configuration index 2 from the configuration list named `myConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.deletelist() (on page 374)
slot[Z].psu[X].configlist.size() (on page 377)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.deletelist()

This function deletes a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].configlist.deletelist("configListName")`

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Details

Use this function to delete a configuration list.

Example

```
slot[1].psu[1].configlist.deletelist("myConfigList")
```

Deletes the configuration list named `myConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)

slot[Z].psu[X].configlist.query()

This function returns a full or partial configuration for the given step.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
config = slot[Z].psu[X].configlist.query("configListName", index)
config = slot[Z].psu[X].configlist.query("configListName", index, diff)
```

config	A table containing all channel attributes or only attributes that are different from the reference configuration at the given index
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number that defines a configuration index in the configuration list
diff	A boolean value. `True` returns only the differences in the selected configuration from the reference configuration. `False` returns the full selected configuration

Details

This function will return a configuration table for the given step. If *diff* is not provided or is set to `false`, the table returned will contain all attributes. If *diff* is `true`, only the attributes that are different than the reference configuration for the named configuration list are returned.

Example

```
config = slot[Z].psu[X].configlist.query("myConfigList", 5)
```

Returns all configuration settings saved in index 5 of the configuration list named `myConfigList`.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.reference() (on page 377)
slot[Z].psu[X].configlist.store() (on page 378)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.recall()

This function recalls and applies the configuration at a specified index to the channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].configlist.recall("configListName", index)`

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a configuration list. These recalled settings are immediately applied to the channel. The active configuration of the instrument channel is modified to match the recalled configuration. A configuration can be called if the channel output is on or off. Each index contains most of the measure and source attributes of the channel.

An error message is generated:

- If you do not specify an index.
- If you specify an invalid index (for example, calling index 3 when there are only two indexes in the configuration list).
- If you try to recall an index from an empty configuration list.

For additional details about the information this command recalls when using a configuration list query command, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].psu[1].configlist.recall("MyConfigList", 5)
```

Recalls and applies configuration index 5 in a configuration list named `MyConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.query() (on page 375)
slot[Z].psu[X].configlist.size() (on page 377)
slot[Z].psu[X].configlist.store() (on page 378)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.reference()

This function returns a table of the reference configuration used when the specified configuration list on the channel was created.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

config = slot[*Z*].psu[*X*].configlist.reference("*configListName*")

config	A table containing the attributes that represent the reference configuration
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Details

Use this function to return the reference configuration used to create the specified configuration list on a channel. This configuration is returned as a Lua table.

Example

```
referenceConfig = slot[1].psu[1].configlist.reference("myConfigList")
```

Returns the reference configuration of `myConfigList` on channel 1 of the module in slot 1 to `referenceConfig`.

Also see

slot[Z].psu[X].configlist.create() (on page 372)

slot[Z].psu[X].configlist.size()

This function returns the number of configuration indexes in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

indexCount = slot[*Z*].psu[*X*].configlist.size("*configListName*")

indexCount	A number that represents the total count of indexes stored in the specified configuration list
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Example

```
print(slot[1].psu[1].configlist.size("testMeasList"))
```

Returns the number of configuration indexes in a configuration list named `testMeasList` on channel 1 of the module installed in slot 1.

Example output:

```
1
```

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.query() (on page 375)
slot[Z].psu[X].configlist.recall() (on page 376)
slot[Z].psu[X].configlist.store() (on page 378)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.store()

This function stores a configuration into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].psu[X].configlist.store("configListName")
slot[Z].psu[X].configlist.store("configListName", index)
slot[Z].psu[X].configlist.store("configListName", index, config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list
config	A table of channel attributes

Details

Use this command to store a set of attributes to a configuration list at the specified index. If no configuration is provided, the active settings on the specified channel and slot are used. If a partial configuration is used, the missing values are taken from the reference configuration for the configuration list.

If the index already contains a configuration, that configuration is overwritten. If the index is the number of steps plus 1 or not specified, the configuration is saved at the end of the list. An error is generated if the index is greater than 1 more than the number of steps already present in the configuration list. To always add the configuration to the end of the configuration list, use `slot[Z].psu[X].configlist.append()`.

Configuration lists are not saved when the instrument is turned off. They are preserved through a reset.

The former functionality of this command is supported but deprecated beginning with mainframe firmware version 1.1.0. Use this command to store a configuration into a named configuration list and `slot[Z].psu[X].configlist.append()` to add steps to an existing configuration list.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].psu[1].configlist.store("MyConfigList")
```

Stores the active settings of the instrument to the end of the configuration list `MyConfigList` on channel 1 of the module installed in slot 1.

```
slot[1].psu[1].configlist.store("MyConfigList", 5)
```

Stores the active settings of the instrument to configuration index 5 in the configuration list `MyConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.append() (on page 371)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.query() (on page 375)
slot[Z].psu[X].configlist.recall() (on page 376)
slot[Z].psu[X].configlist.reference() (on page 377)
slot[Z].psu[X].configlist.size() (on page 377)
slot[Z].psu[X].configlist.table() (on page 379)

slot[Z].psu[X].configlist.table()

This function returns the names of configuration lists stored on the channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
configListTable = slot[Z].psu[X].configlist.table()
```

configListTable	A table that contains the names of the configuration lists stored on the specified slot and channel
Z	Module slot number
X	Module channel number

Details

This function returns a Lua table that contains the names of configuration lists that are stored on the specified channel.

To see the names of the configuration lists that have been created, use either of the code examples under Example 1 and Example 2. If no configuration lists exist, the table is empty.

Example 1

```
-- Create configuration lists.
slot[1].psu[1].configlist.create("configlist1")
slot[1].psu[1].configlist.create("configlist2")
-- Retrieve the names of the lists.
myConfigLists = slot[1].psu[1].configlist.table()
for i, name in pairs(myConfigLists) do
    print(name)
end
```

Create configuration lists.

Retrieve the names of all configuration lists:

```
configlist1
configlist2
```

Example 2

```
-- Create configuration lists.
slot[1].psu[1].configlist.create("configlist1")
slot[1].psu[1].configlist.create("configlist2")
-- Retrieve the names of the lists.
printbuffer(1, table.getn(myConfigLists), myConfigLists)
end
```

Create configuration lists.

Retrieve the names of all configuration lists:

```
configlist1
configlist2
```

Also see

Configuration lists (on page 50)
slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].psu[X].configlist.query() (on page 375)
slot[Z].psu[X].configlist.recall() (on page 376)
slot[Z].psu[X].configlist.size() (on page 377)
slot[Z].psu[X].configlist.store() (on page 378)

slot[Z].psu[X].defbufferY

This attribute contains the default reading buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle slot[Z].smu[X].defbuffer1.clear()	Not applicable	Not applicable

Usage

bufferContents = slot[*Z*].psu[*X*].defbuffer*Y*
bufferContents = slot[*Z*].psu[*X*].defbuffer*Y*[*N*]

bufferContents	The readings in the default reading buffer
Z	Module slot number
X	Module channel
Y	Default buffer number: 1 or 2
N	Index of the reading in the buffer

Details

Each PSU channel contains two default reading buffers: slot[*Z*].psu[*X*].defbuffer1 and slot[*Z*].psu.[*X*].defbuffer2.

All routines that return measurements can also store them in either reading buffer. Overlapped measurements are always stored in a reading buffer. Synchronous measurements return either a single-point measurement or are stored in a reading buffer if they are passed to the measurement command.

Example

```
slot[1].psu[2].measure.count = 10
slot[1].psu[2].measure.v(slot[1].psu[2].defbuffer1)
printbuffer(1, slot[1].psu[2].defbuffer1.n, slot[1].psu[2].defbuffer1.readings)
```

Stores 10 voltage measurements in `defbuffer1` of channel 2 of the module installed in slot 1 and returns the readings. The use of `defbuffer1.n (bufferVar.n)` indicates that the instrument should output all readings in the reading buffer. Example output:

```
-7.70786e-04, -6.81580e-04, -7.87006e-04, -7.70786e-04, -7.46457e-04, -7.70786e-04
, -6.81580e-04, -7.95115e-04, -7.70786e-04, -7.78896e-04
```

Also see

Reading buffers (on page 96)

slot[Z].psu[X].makebuffer()

This function creates a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar = slot[*Z*].psu[*X*].makebuffer(*bufferSize*)

bufferVar	The variable name of the new reading buffer
Z	Module slot number
X	Module channel number
bufferSize	The size of the reading buffer

Details

Reading buffers can be created and allocated dynamically using this function. Use *bufferSize* to designate the number of readings the buffer can store.

An error is returned if *bufferSize* exceeds the maximum storage capacity of the module. The available storage capacity varies and depends on how much memory is consumed by other reading buffers, configuration lists, trigger models, and sweep configurations.

Dynamically allocated reading buffers can be used interchangeably with the dedicated slot[*Z*].psu[*X*].defbuffer*Y* buffers.

To delete a reading buffer, set all references to the reading buffer equal to `nil`, then run the garbage collector.

Example

```
mybuffer2 = slot[1].psu[2].makebuffer(200)
```

Creates a reading buffer (`mybuffer2`) that holds 200 readings for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].defbufferY (on page 380)

slot[Z].psu[X].measure.aperture

This attribute contains the measurement aperture for a channel in seconds.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	slot[Z].psu[X].measure.rate	Not applicable	6.66670e−02 (60 Hz line) 8e-02 (50 Hz line)

Usage

```
value = slot[Z].psu[X].measure.aperture
```

value	The measure aperture in seconds for the specified line frequencies: ▪ Rate set to RATE_NORMAL: 6.66670e−02 (60 Hz), 8e-2 (50 Hz) ▪ Rate set to RATE_FAST: 1.6667e-3 (60 Hz and 50 Hz)
Z	Module slot number
X	Module channel number

Details

The aperture and NPLC attributes cannot be set directly. These attributes report the present measure rate in seconds or NPLC. The measure rate is set using the `slot[Z].psu[X].measure.rate` attribute.

Example

```
print(slot[1].psu[2].measure.aperture)
```

Returns the present measure aperture in seconds for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].measure.nplc (on page 383)
slot[Z].psu[X].measure.rate (on page 385)

slot[Z].psu[X].measure.count

This attribute sets the number of measurements that are made when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	1

Usage

```
count = slot[Z].psu[X].measure.count
slot[Z].psu[X].measure.count = count
```

count	Number of measurements
Z	Module slot number
X	Module channel number

Details

This attribute controls the number of measurements made and stored when a measurement is requested to be stored in a reading buffer. If a reading buffer is not specified, this attribute is ignored and one measurement is made.

If the count exceeds the remaining capacity of the reading buffer, no measurements are made.

If the count is set to a value greater than 1, any measurement delay set by `slot[Z].psu[X].measure.delay` only occurs before the first measurement.

Example

```
slot[1].psu[1].measure.count = 100
```

Sets 100 measurements to be made on channel 1 of the module installed in slot 1.

Also see

slot[Z].psu[X].measure.Y() (on page 389)

slot[Z].psu[X].measure.nplc

This attribute contains the measurement rate in number of power line cycles.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	slot.[Z].psu[X].measure.rate	Not applicable	4

Usage

rate = `slot[Z].psu[X].measure.nplc`

rate	Power line cycles: 4 (`RATE_NORMAL`) or 0.1 (`RATE_FAST`)
Z	Module slot number
X	Module channel number

Details

The aperture and NPLC attributes cannot be set directly. These attributes report the present measure rate in seconds or NPLC. The measure rate is set using the `slot[Z].psu[X].measure.rate` attribute.

This attribute returns the measurement rate as the number of power line cycles (NPLC) for the channel. Each PLC for 60 Hz is 16.67 ms (1/60) and each PLC for 50 Hz is 20 ms (1/50).

Example

```
slot[1].psu[2].measure.rate = slot[1].psu[2].RATE_FAST
NPLC1 = slot[1].psu[2].measure.nplc
print(NPLC1)
```

Reads the measurement aperture in NPLCs for channel 2 of the module installed in slot 1 and prints the value of NPLC1. Example output:

```
1.00002e-01
```

Also see

slot[Z].psu[X].measure.aperture (on page 382)
slot.[Z].psu[X].measure.rate (on page 385)

slot[Z].psu[X].measure.overlappedY()

This function starts an asynchronous (background) measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
slot[Z].psu[X].measure.overlappedY(rbuffer)
slot[Z].psu[X].measure.overlappediv(ibuffer, vbuffer)
```

Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
rbuffer	A reading buffer object where readings are stored
ibuffer	A reading buffer object where current readings are stored
vbuffer	A reading buffer object where voltage readings are stored

Details

This function starts a measurement and immediately returns the readings. The measurements are stored in a reading buffer, along with any ancillary information also acquired. If the instrument is configured to return multiple readings, the readings become available as they are made.

Measurements are in the following units: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

The `slot[Z].psu[X].measure.overlappediv()` function stores current readings in *ibuffer* and voltage readings in *vbuffer*.

This function is an overlapped command. Script execution continues while measurements are made in the background. Attempts to access result values that have not yet been generated cause the script to wait for the data to become available. The `waitcomplete()` function can also be used to wait for the measurements to complete before continuing with the script.

Any data in a reading buffer is deleted before recording any measurements unless the reading buffer has been configured to append data.

Overlapped measure commands cannot be used with the trigger model.

Example

```
slot[1].psu[1].measure.overlappedv(slot[1].psu[1].defbuffer1)
```

Starts background voltage measurements for channel 1 of the module installed in slot 1.

Also see

slot[Z].psu[X].defbufferY (on page 380)
slot[Z].psu[X].measure.Y() (on page 389)

slot[Z].psu[X].measure.rangeY

This attribute contains the positive full-scale value of the measurement range for voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Voltage: 50 V Current: 5 A

Usage

```
rangeValue = slot[Z].psu[X].measure.rangeY
```

rangeValue	Voltage range: 50 V Current range: 5 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This attribute indicates the range used to measure voltage or current. The MPSU50-2ST has only one voltage range and one current range.

Example

```
print(slot[1].psu[2].measure.rangev)
```

Displays the present voltage measurement range for channel 2 of the module installed in slot 1.

Also see

None

slot[Z].psu[X].measure.rate

This attribute configures the measure rate for a PSU channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].psu[X].RATE_NORMAL

Usage

```
measureRate = slot[Z].psu[X].measure.rate
slot[Z].psu[X].measure.rate = measureRate
```

measureRate	The predefined measure rate, which is one of the following values: ▪ `slot[Z].psu[X].RATE_FAST` ▪ `slot[Z].psu[X].RATE_NORMAL`
Z	Module slot number
X	Module channel number

Details

This attribute configures the measure rate for the channel to one of the predefined rates.

Setting this attribute changes the measurement aperture of the analog-to-digital converter. The measurement aperture is the amount of time that the input signal is measured. The fast reading rate increases reading noise. The normal reading rate takes longer but has the lowest reading noise.

This setting changes the reported NPLC and aperture attributes to match the predefined rate.

Example

```
slot[1].psu[2].measure.rate = slot[1].psu[2].RATE_FAST
```

Sets the measure rate of the power supply to the `FAST` setting for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].measure.aperture (on page 382)
slot[Z].psu[X].measure.nplc (on page 383)

slot[Z].psu[X].measure.rel.enableY

This attribute turns relative measurements on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].psu[X].DISABLE

Usage

```
relEnable = slot[Z].psu[X].measure.rel.enableY
slot[Z].psu[X].measure.rel.enableY = relEnable
```

relEnable	Enable or disable relative measurements: ▪ Disables relative measurements: `slot[Z].psu[X].DISABLE` ▪ Enables relative measurements: `slot[Z].psu[X].ENABLE`
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power

Details

This attribute enables or disables relative measurements. When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value specified by `slot[Z].psu[X].measure.rel.levelY`.

Each returned measured relative reading is the result of the following:

Relative reading = Actual measured reading − Relative offset value

Example

```
slot[1].psu[2].measure.rel.enablev = slot[1].psu[2].ENABLE
```

Enables relative voltage measurements for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].measure.rel.levelY (on page 387)

slot[Z].psu[X].measure.rel.levelY

This attribute specifies the offset value for relative measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0

Usage

```
relValue = slot[Z].psu[X].measure.rel.levelY
slot[Z].psu[X].measure.rel.levelY = relValue
```

relValue	Relative measurement offset value
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power

Details

When relative measurements are enabled (`slot[Z].psu[X].measure.rel.enableY`), all subsequent measured readings are offset by the value of this attribute.

Each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading − Relative offset value

Example

```
slot[1].psu[2].measure.rel.enablev = slot[1].psu[2].ENABLE
slot[1].psu[2].measure.rel.levelv = slot[1].psu[2].measure.v()
```

Enable the relative offset for voltage measurement for channel 2 of the module installed in slot 1.

Make a voltage measurement and then use the reading as the relative offset value.

Also see

slot[Z].psu[X].measure.rel.enableY (on page 386)

slot[Z].psu[X].measure.tempcomp

This attribute turns temperature compensation on or off for the current measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index Calibration	Configuration list index	slot[*Z*].psu[*X*].ENABLE

Usage

```
status = slot[Z].psu[X].measure.tempcomp
slot[Z].psu[X].measure.tempcomp = status
```

status	Temperature compensation control setting: ▪ Disable: `slot[Z].psu[X].DISABLE` ▪ Enable: `slot[Z].psu[X].ENABLE`
Z	Module slot number
X	Module channel number

Details

Temperature compensation improves accuracy for current measurements by accounting for the temperature coefficient of the current measure circuitry.

When temperature compensation is enabled, the measure rate for current measurements is reduced because the analog-to-digital converter performs an additional temperature measurement. This attribute does not affect voltage measurements or current source accuracy.

NOTE: This setting must be disabled when making measurements for current calibration.

Example

```
slot[1].psu[2].measure.tempcomp = slot[1].psu[2].DISABLE
```

Disables temperature compensation for channel 2 of the module installed in slot 1.

Also see

None

slot[Z].psu[X].measure.Y()

This function makes one or more measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = slot[Z].psu[X].measure.Y()
reading = slot[Z].psu[X].measure.Y(readingBuffer)
reading, compValue = slot[Z].psu[X].measure.i()
reading, compValue = slot[Z].psu[X].measure.i(readingBuffer)
iReading, vReading = slot[Z].psu[X].measure.iv()
iReading, vReading = slot[Z].psu[X].measure.iv(iReadingBuffer, vReadingBuffer)
```

reading	The last or only measured value returned; see **Details**
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
readingBuffer	The reading buffer; can be a user-defined or default reading buffer
compValue	The compensation value used to calibrate current measure
iReading	The last or only current measurement value returned
vReading	The last or only voltage measurement value returned
iReadingBuffer	Reading buffer where current readings are stored; can be a user-defined buffer or a default reading buffer
vReadingBuffer	Reading buffer where voltage readings are stored; can be a user-defined buffer or a default reading buffer

Details

This function initiates one or more synchronous measurements based on the parameters specified above.

Measurements are in the following units of measure:

- `v` = volts
- `i` = amperes
- `r` = ohms
- `p` = watts

If you use this function without specifying a reading buffer, it makes one measurement and returns that measurement as *reading*. To use the additional information that is acquired while making a measurement or to return multiple readings, specify a reading buffer. If the instrument is configured to return multiple readings for a measurement and *readingBuffer* is specified, all readings are available in *readingBuffer*, but only the last measurement is returned as *reading*.

The `slot[Z].psu[X].measure.iv()` function returns the last actual current measurement and voltage measurement as *iReading* and *vReading*, respectively. Additionally, it can store current and voltage readings if buffers are provided (*iReadingBuffer* and *vReadingBuffer*).

The `slot[Z].psu[X].measure.count` attribute determines how many measurements are performed. When a reading buffer is used, it also determines the number of readings to store in the buffer. If a reading buffer is not specified, the PSU ignores the `slot[Z].psu[X].measure.count` attribute and only makes one measurement.

The *readingBuffer* is cleared before making any measurements unless the buffer is configured to append data.

Example

```
voltageValue = slot[1].psu[2].measure.v()
```

Measures the voltage on channel 2 of the module installed in slot 1 and stores it in the `voltageValue` variable.

Also see

slot[Z].psu[X].measure.count (on page 382)
slot[Z].psu[X].measure.rel.enableY (on page 386)

slot[Z].psu[X].overlapped

This attribute contains the state of the overlapped mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].psu[X].DISABLE

Usage

```
overlappedMode = slot[Z].psu[X].overlapped
slot[Z].psu[X].overlapped = overlappedMode
```

overlappedMode	Disable overlapped commands: `slot[Z].psu[X].DISABLE` Enable overlapped commands: `slot[Z].psu[X].ENABLE`
Z	Module slot number
X	Module channel number

Details

When this attribute is set to disable, both sequential and measure overlapped commands proceed as in the order they are sent. For sequential commands, a command is processed fully before the next one is executed.

When this attribute is set to enable, a command returns as soon as it is able, even if it has not been reflected in the hardware. For example, the command for setting the source voltage level returns to Lua so it can process the next command, even if the instrument has not yet transitioned to the new voltage level.

Not all commands can be overlapped. If they can be, the command gets added to the processing queue of the channel and execute in order. Channels process their respective queue in parallel with other channels. If the command cannot be overlapped, the PSU inserts a `waitcomplete()` between commands. For example, if a `print(slot[Z].psu[X].source.rangev))` command is executed after a `slot[Z].psu[X].source.levelv = newLevel` command, the PSU completes the level change before printing the source range as if there were a `waitcomplete()` command executed between them.

This only applies to commands that specify settings. A command that retrieves a setting or a measure command waits for commands to complete before executing. Note that measure commands do not wait for commands to fully complete on a different channel.

This attribute is useful when performing operations on multiple channels. Changing the voltage level on channel 1 and channel 2 with this setting enabled should result in those changes occurring with less time between than if this setting was disabled.

Overlapped measure commands cannot be used with the trigger model.

Example

```
slot[1].psu[1].overlapped = slot[1].psu[1].ENABLE
slot[2].psu[2].overlapped = slot[2].psu[2].ENABLE
slot[1].psu[1].source.levelv = 5
slot[2].psu[2].source.levelv = 5
slot[1].psu[1].source.output = slot[1].psu[1].ON
slot[2].psu[2].source.output = slot[2].psu[2].ON
```

PSU channel 2 begins to turn its output on while channel 1 is still completing its output on command.

Also see

None

slot[Z].psu[X].reset()

This function turns off the output and resets the attributes that begin with `psu[X].` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].reset()`

Z	Module slot number
X	Module channel number

Details

This function turns off the output and returns the specified PSU to its default settings.

Example

```
slot[1].psu[1].reset()
```

Turns off the output and resets channel 1 of the module installed in slot 1 to its default settings.

Also see

None

slot[Z].psu[X].source.constantcurrent

This attribute indicates whether the power supply is operating in a constant-voltage or constant-current state.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

```
value = slot[Z].psu[X].source.constantcurrent
```

value	The state of the power supply: ▪ Constant-current state: `true` ▪ Constant-voltage state: `false`
Z	Module slot number
X	Module channel number

Details

The power supply module features a constant voltage and constant current automatic crossover. This feature permits continuous operation in the transition from a constant-voltage state to a constant-current state as the load changes. This attribute allows you to check which state is active.

Example

```
print(slot[1].psu[2].source.constantcurrent)
```

Returns whether the source connected to channel 2 of the module installed in slot 1 is in the constant-voltage or constant-current state.

Also see

None

slot[Z].psu[X].source.levelv

This attribute configures the voltage source level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0

Usage

```
sourceLevel = slot[Z].psu[X].source.levelv
slot[Z].psu[X].source.levelv = sourceLevel
```

sourceLevel	Voltage source level in volts: 0 to ±50.1
Z	Module slot number
X	Module channel number

Details

This attribute configures the output level of the voltage source.

If the output is on, the new setting goes in effect immediately. If the output is off, the voltage level is sourced when the output is turned on. The output voltage changes according to the slew rate set by `slot[Z].psu[X].source.slewrate.v`.

The sign of *`sourceLevel`* dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

The `slot[Z].psu[X].reset()` function sets the source levels to 0 V and 0 A.

Example

```
slot[1].psu[2].source.levelv = 3.5
```

Sets the source level to 3.5 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].reset() (on page 391)
slot[Z].psu[X].source.slewrate.v (on page 402)

slot[Z].psu[X].source.limiti

This attribute limits the current output of the power supply.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0.5 A

Usage

```
value = slot[Z].psu[X].source.limiti
slot[Z].psu[X].source.limiti = value
```

value	The current limit for the source in amps: 0.01 to 5.1
Z	Module slot number
X	Module channel number

Details

If the current exceeds this value, the voltage level is limited so that the current remains below the set limit.

For example, assume you set a current limit of 1 A and connect a 5 ohm resistor to the power supply output. If you set the voltage level to 2 V, 2 V/ 5 Ω = 400 mA. Since this is less than 1 A, the power supply outputs 2 V as requested. If you change the voltage level to 6 V, 6 V/5 Ω = 1.2 A. Since this is more than 1 A, the current limit takes effect and limits the output to 1 A. In this example, the output voltage is 5 V even though the level was set to 6 V.

The current limit affects the slew rate. Set the current limit to greater than or equal to 500 mA to achieve the specified slew rate.

Example

```
slot[1].psu[2].source.limiti = 4.8
```

Sets source current limit of channel 2 of the module installed in slot 1 to 4.8 A.

Also see

slot[Z].psu[X].source.slewrate.v (on page 402)

slot[Z].psu[X].source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset	Not applicable	slot[Z].psu[X].OFF

Usage

```
value = slot[Z].psu[X].source.output
slot[Z].psu[X].source.output = value
```

value	State of the source output: ▪ Enable: `slot[Z].psu[X].ON` ▪ Disable: `slot[Z].psu[X].OFF`
Z	Module slot number
X	Module channel number

Details

When this attribute is read, it returns the output state of the source. When this attribute is set, it switches the output of the source on or off. The output voltage changes according to the slew rate set by `slot[Z].psu[X].source.slewrate.v`.

Set this attribute to `slot[Z].psu[X].OFF` to place the output in a high-impedance state.

The source output setting is not stored in configuration lists.

Example

```
slot[1].psu[2].source.output = slot[1].psu[2].ON
```

Turns on the output for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.slewrate.v (on page 402)

slot[Z].psu[X].source.protect.enablei

This attribute enables or disables overcurrent protection.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[*Z*].psu[*X*].ENABLE

Usage

```
protectEnabled = slot[Z].psu[X].source.protect.enablei
slot.[Z].psu[X].source.protect.enablei = protectEnabled
```

protectEnabled	Enable or disable overcurrent protection: ▪ Disable: `slot[Z].psu[X].DISABLE` ▪ Enable: `slot[Z].psu[X].ENABLE`
Z	Module slot number
X	Module channel number

Details

When overcurrent protection is enabled, the power supply turns off the output of the protected channel when the output current exceeds the protection level attribute.

When the protection trips, the slew rate limit setting is ignored.

Example

```
slot[1].psu[2].source.protect.enablei = slot[1].psu[2].DISABLE
```

Disables overcurrent protection for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.enablev (on page 396)
slot[Z].psu[X].source.protect.leveli (on page 397)
slot[Z].psu[X].source.protect.levelv (on page 398)
slot[Z].psu[X].source.protect.trippedi (on page 399)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].psu[X].source.protect.enablev

This attribute enables or disables the output overvoltage protection.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].psu[X].ENABLE

Usage

```
protectEnabled = slot[Z].psu[X].source.protect.enablev
slot[Z].psu[X].source.protect.enablev = protectEnabled
```

protectEnabled	The state of overvoltage protection: ▪ Disable: `slot[Z].psu[X].DISABLE` ▪ Enable: `slot[Z].psu[X].ENABLE`
Z	Module slot number
X	Module channel number

Details

When overcurrent protection is enabled and the output current exceeds the set current protection level, the power supply turns off the output of the protected channel.

When the protection trips, the slew rate limit setting is ignored.

Example

```
slot[1].psu[2].source.protect.enablev = slot[1].psu[2].DISABLE
```

Disables overvoltage protection for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.enablei (on page 395)
slot[Z].psu[X].source.protect.leveli (on page 397)
slot[Z].psu[X].source.protect.levelv (on page 398)
slot[Z].psu[X].source.protect.trippedi (on page 399)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].psu[X].source.protect.leveli

This attribute sets the level for the output overcurrent protection.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	5.5 A

Usage

```
protectLevel = slot[Z].psu[X].source.protect.leveli
slot[Z].psu[X].source.protect.leveli = protectLevel
```

protectLevel	The current protection level in amps: 0.25 to 5.5
Z	Module slot number
X	Module channel number

Details

If the output current rises above this level, the power supply immediately turns off the output of the protected channel.

When the protection trips, the slew rate limit setting is ignored.

This attribute is in effect when `slot[Z].psu[X].source.protect.enablei` is enabled.

When overcurrent protection is used in a test sequence, set it before turning the source on.

Example

```
slot[1].psu[2].source.protect.leveli = 1.0
```

Sets the output overcurrent protection level to 1.0 A for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.enablei (on page 395)
slot[Z].psu[X].source.protect.levelv (on page 398)
slot[Z].psu[X].source.protect.trippedi (on page 399)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].psu[X].source.protect.levelv

This attribute configures the output overvoltage protection level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	55 V

Usage

```
protectLevel = slot[Z].psu[X].source.protect.levelv
slot[Z].psu[X].source.protect.levelv = protectLevel
```

protectLevel	0.1 V to 55 V
Z	Module slot number
X	Module channel number

Details

If the voltage on the output terminals goes above this level, the power supply immediately turns off the output of the protected channel.

When the protection trips, the slew rate limit setting is ignored.

When overvoltage protection is used in a test sequence, set it before turning the source on.

This attribute is in effect when `slot[Z].psu[X].source.protect.enablev` is enabled.

Example

```
slot[1].psu[2].source.protect.levelv = 10.0
```

Sets the output overvoltage protection level to 10.0 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.enablev (on page 396)
slot[Z].psu[X].source.protect.leveli (on page 397)
slot[Z].psu[X].source.protect.trippedi (on page 399)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].psu[X].source.protect.trippedi

This attribute indicates if the overcurrent protection circuit was tripped.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

tripped = slot[*Z*].psu[*X*].source.protect.trippedi

tripped	The overcurrent protection status: ▪ The overcurrent protection circuit tripped: `true` ▪ The protection circuit is not tripped: `false`
Z	Module slot number
X	Module channel number

Details

This attribute indicates whether the overcurrent protection circuit was actuated. When the overcurrent protection circuit is actuated, the output of the protected channel is turned off.

The status is reset to `false` and the protection circuit is reset when:

- The output is turned on
- When the mainframe is reset
- When the module is reset

Example

```
print(slot[1].psu[2].source.protect.trippedi)
```

Determines if the overcurrent protection circuit was activated for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.leveli (on page 397)
slot[Z].psu[X].source.protect.levelv (on page 398)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].psu[X].source.protect.trippedv

This attribute indicates if the overvoltage protection circuit was tripped.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

tripped = slot[*Z*].psu[*X*].source.protect.trippedv

tripped	The overvoltage protection status: ▪ The overvoltage protection circuit was tripped: `true` ▪ The protection circuit is not tripped: `false`
Z	Module slot number
X	Module channel number

Details

This attribute indicates whether the overvoltage protection circuit was activated. If the overvoltage protection circuit was activated, the output of the protected channel is turned off.

The status is reset to `false` and the protection circuit is reset when

- The output is turned on
- The mainframe is reset
- The module is reset

Example

```
print(slot[1].psu[2].source.protect.trippedv)
```

Reports if the overvoltage protection circuit was activated for channel 2 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.protect.leveli (on page 397)
slot[Z].psu[X].source.protect.levelv (on page 398)
slot[Z].psu[X].source.protect.trippedi (on page 399)

slot[Z].psu[X].source.rangev

This attribute contains the value of the source range for voltage.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	50 V

Usage

value = slot[*Z*].psu[*X*].source.rangev

value	Voltage range: 50 V
Z	Module slot number
X	Module channel number

Details

This setting contains the fixed range setting for the voltage source.

Example

```
print(slot[1].psu[2].source.rangev)
```

Returns the source range for channel 2 of the module installed in slot 1.

Also see

None

---

slot[Z].psu[X].source.slewrate.fallv

This attribute sets the falling voltage source slew rate.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Module reset	Configuration list index	10 kV per second

Usage

```
value = slot[Z].psu[X].source.slewrate.fallv
slot[Z].psu[X].source.slewrate.fallv = value
```

value	The slew rate in volts per second, 1 V to 10 kV
Z	Module slot number
X	Module channel number

Details

This attribute configures the falling slew rate of the voltage source in volts per second. The rate can be set to any value between 1 V per second and 10 kV per second.

Example

```
slot[1].psu[1].source.slewrate.fallv = 5.0
```

This example sets the falling slew rate to 5 V per second for channel 1 of the module in slot 1.

Also see

slot[Z].psu[X].source.levelv (on page 392)
slot[Z].psu[X].source.output (on page 394)
slot[Z].psu[X].source.slewrate.risev (on page 402)
slot[Z].psu[X].source.slewrate.v (on page 402)

slot[Z].psu[X].source.slewrate.risev

This attribute sets the rising voltage source slew rate.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Module reset	Configuration list index	10 kV per second

Usage

```
value = slot[Z].psu[X].source.slewrate.risev
slot[Z].psu[X].source.slewrate.risev = value
```

value	The slew rate in volts per second, 1 V to 10 kV
Z	Module slot number
X	Module channel number

Details

This attribute configures the rising slew rate of the voltage source in volts per second. The rate can be set to any value between 1 V per second and 10 kV per second.

Example

```
slot[1].psu[1].source.slewrate.risev = 5.0
```

This example sets the rising slew rate to 5 V per second for channel 1 of the module in slot 1.

Also see

slot[Z].psu[X].source.levelv (on page 392)
slot[Z].psu[X].source.output (on page 394)
slot[Z].psu[X].source.slewrate.fallv (on page 401)
slot[Z].psu[X].source.slewrate.v (on page 402)

slot[Z].psu[X].source.slewrate.v

This attribute sets the rising and falling voltage source slew rate.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Module reset	Configuration list index	10 kV per second

Usage

```
value = slot[Z].psu[X].source.slewrate.v
slot[Z].psu[X].source.slewrate.v = value
```

value	The slew rate in volts per second, 1 V to 10 kV
Z	Module slot number
X	Module channel number

Details

This attribute configures the slew rate of the voltage source in volts per second. The slew rate can be set to any value between 1 V per second and 10 kV per second. Reading this attribute returns the `slot[Z].psu[X].source.slewrate.risev` setting.

The maximum slew rate is achieved when stepping from 0 V to ± full scale with a current limit >500 mA and a load >65 ohms.

Actual slew rate is limited by current limit, load, and voltage level. If the current drawn by the load is near the current limit, the slew rate is reduced to prevent overshoot. Increase the current limit to improve slew rate performance. Maximum slew rate performance is achieved with current limits greater than 500 mA. For more information, see the *MPSU50-2ST Power Supply Reference Manual*

This attribute adjusts both the `slot[Z].psu[X].source.slewrate.risev` and `slot[Z].psu[X].source.slewrate.fallv` settings.

This command replaces `slot[Z].psu[X].source.slewratev`, which is deprecated.

Example

```
slot[1].psu[1].source.slewrate.v = 5.0
```

This example sets the rising and falling slew rate to 5 V per second for channel 1 of the module installed in slot 1.

Also see

slot[Z].psu[X].source.levelv (on page 392)
slot[Z].psu[X].source.limiti (on page 393)
slot[Z].psu[X].source.output (on page 394)
slot[Z].psu[X].source.slewrate.fallv (on page 401)
slot[Z].psu[X].source.slewrate.risev (on page 402)

slot[Z].psu[X].trigger.measure.Y()

This function configures the measurements to be made during trigger model operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].psu[X].trigger.measure.Y(readingBuffer)
slot[Z].psu[X].trigger.measure.iv(currentBuffer, voltageBuffer)
```

Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
readingBuffer	The reading buffer; can be a user-defined or default reading buffer
currentBuffer	Reading buffer where current readings are stored; can be a user-defined buffer or a default reading buffer
voltageBuffer	Reading buffer where voltage readings are stored; can be a user-defined buffer or a default reading buffer

Details

This function configures the measurement subsystem for one or more synchronous measurements to be made while a trigger model is executed.

Measurements are in the following units of measure:

- `v` = volts
- `i` = amperes
- `r` = ohms
- `p` = watts

This command only configures the measure functions and reading buffers to be used during a trigger model; it does not cause measurements to be made. To make measurements, you must add a measure block to the trigger model using the `slot[Z].psu[X].trigger.model.addblock.measure()` command.

The `slot[Z].psu[X].measure.count` attribute is not used by the measure subsystem when a trigger model is executed. Instead, the measure count is passed in through the measure block command.

The reading buffers are cleared when `slot[Z].trigger.model.initiate()` is called unless the buffer is configured to append data. Reading buffers are not optional for this command.

The module only retains the last call to any one of these functions and only that measurement is made.

Example

```
--- Configure PSU Ch1 to measure voltage and save the result in a reading buffer.
ReadingBuffer = slot[1].psu[1].makebuffer(10)
slot[1].psu[1].trigger.measure.v(ReadingBuffer)
--- Set the initial voltage output for PSU Ch1 to 2 V.
slot[1].psu[1].source.levelv = 2
--- Delete old trigger model instances and then create new one.
slot[1].trigger.model.delete("TM1")
slot[1].trigger.model.create("TM1")
--- Make the voltage measurement 10 times.
slot[1].trigger.model.addblock.source.output("TM1", "OutputOn", 1, 1)
slot[1].trigger.model.addblock.measure("TM1", "MeasureVoltage", 1, 10)
slot[1].trigger.model.addblock.source.output("TM1", "OutputOff", 1, 0)
--- Execute the trigger model.
slot[1].trigger.model.initiate("TM1")
waitcomplete()
-- Use printbuffer to return the 10 measurements.
printbuffer(1, ReadingBuffer.n, ReadingBuffer)
```

A reading buffer is configured for voltage measurements and a measure block captures measurements for the module installed in slot 1.

Also see

slot[Z].trigger.model.addblock.measure() (on page 590)

slot[Z].psu[X].trigger.source.linearY()

This function configures a linear sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].trigger.source.linearY(start, end, points)`

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage
start	Start source level
end	End source level
points	Number of points used to calculate the step size

Details

This command sets up a linear sweep by configuring the start source level, end source level, and the number of steps to increase or decrease from the starting source level to the ending source level. When you use a linear staircase sweep, the voltage source increases or decreases in fixed steps. Each point is equally spaced between the start and stop.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The PSU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

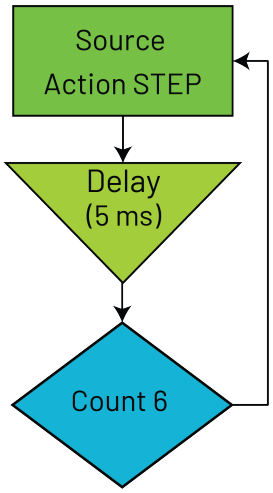
- `slot[Z].psu[X].trigger.source.linearY()`
- `slot[Z].psu[X].trigger.source.listY()`
- `slot[Z].psu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].psu[1].trigger.source.linearv(1, 5, 6)
```

Configure a six-point linear sweep from 1 V to 5 V on channel 1 in slot 1.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.set() (on page 606)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

slot[Z].psu[X].trigger.source.listY()

This function configures a list sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].psu[X].trigger.source.listY(sourceLevels)`

Z	Module slot number
X	Module channel number
Y	The voltage source function; `v`
sourceLevels	List of source levels such as `{1, 2, 3}`; see **Details**

Details

This function configures a sweep from a predetermined list passed through the *sourceLevels* parameter. The number of elements in *sourceLevels* does not set the number of steps in a sweep. Instead, it provides the list of source values to be used by a subsequent sweep.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The PSU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

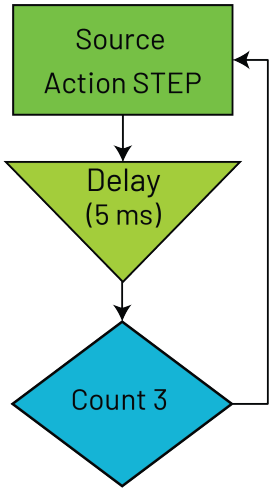
- `slot[Z].psu[X].trigger.source.linearY()`
- `slot[Z].psu[X].trigger.source.listY()`
- `slot[Z].psu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].psu[1].trigger.source.listv({1, 2, 3})
```

Configure a list sweep using specific source levels 1 V, 2 V, and 3 V on channel 1 in slot 1.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step (on page 610)

slot[Z].psu[X].trigger.source.logY()

This function configures a logarithmic sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].psu[X].trigger.source.logY(start, end, points)
slot[Z].psu[X].trigger.source.logY(start, end, points, asymptote)
```

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage
start	Start source level
end	End source level
points	Number of points used to calculate the step size
asymptote	The asymptotic offset value

Details

This command sets up a logarithmic sweep by configuring the start source level, end source level, and the number of points to increase or decrease from the starting source level to the ending source level.

The *points* parameter does not set the number of steps in a sweep. Instead, it is used to generate a list of source values to be used by a subsequent sweep.

A logarithmic staircase sweep is similar to a linear staircase sweep. In a logarithmic sweep, however, the steps are scaled logarithmically.

The *asymptote* parameter customizes the inflection and offset of the source value curve. This allows log sweeps to cross zero. Setting this parameter to zero provides a conventional logarithmic sweep. The *asymptote* value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep.

The *asymptote* value must be outside the range defined by the start and end values. It must not be equal to or between the start and end values.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The PSU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

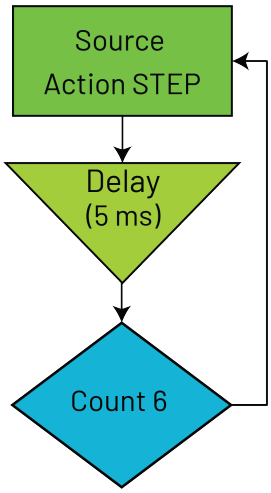
- `slot[Z].psu[X].trigger.source.linearY()`
- `slot[Z].psu[X].trigger.source.listY()`
- `slot[Z].psu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].psu[1].trigger.source.logv(1, 5, 6)
```

Configure a six-point logarithmic sweep from 1 V to 5 V on channel 1 in slot 1.

Trigger model example



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.set() (on page 606)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

slot[Z].psu[X].waitcomplete()

This function waits for all previously started overlapped commands to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`slot[Z].psu[X].waitcomplete()`

Z	Module slot number
X	Module channel number

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `slot[Z].psu[X].waitcomplete()` command is a sequential command that can be used as a synchronization point between channels. It does not return until all previously started overlapped commands on the specified channel of a module are complete. This effectively suspends the execution of all commands following `slot[Z].psu[X].waitcomplete()` until then.

Example

```
slot[1].psu[1].waitcomplete()
```

Suspends execution of the TSP script until any overlapped commands complete.

Also see

waitcomplete() (on page 347)

slot[Z].revision

This attribute stores the firmware revision number of the module in the specified slot of the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Firmware update	Nonvolatile memory	Not applicable

Usage

revisionNumber = slot[*Z*].revision

revisionNumber	The module firmware revision number
Z	Module slot number

Example

```
print(slot[1].revision)
```

Returns the firmware revision number of the module installed in slot 1.

Also see

None

slot[Z].serialno

This attribute stores the serial number of the module installed in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

serialno = slot[*Z*].serialno

serialno	The serial number of the module
Z	Module slot number

Example

```
print(slot[1].serialno)
```

Returns the serial number of the module installed in slot 1.

Also see

None

slot[Z].status.*

This attribute contains the registers of the status register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—-	—	—	—
.condition (R)	Yes	Not applicable	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	137 (all bits set)

Usage

```
measurementRegister = slot[Z].status.condition
measurementRegister = slot[Z].status.enable
measurementRegister = slot[Z].status.event
measurementRegister = slot[Z].status.ntr
measurementRegister = slot[Z].status.ptr
slot[Z].status.enable = measurementRegister
slot[Z].status.ntr = measurementRegister
slot[Z].status.ptr = measurementRegister
```

measurementRegister	The status of the slot status summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the slot status summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 129 is returned for the enable register. The binary equivalent is `0000 0000 1000 0001`. This value indicates that bits B0 (MSB) and B7 (OSB) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	`slot[Z].status.MSB` Set bit is a summary of the `slot[Z].status.measurement` register. Bit B0 decimal value: 1
B1 to B2	Not used
B3	`slot[Z].status.QSB` Set bit is a summary of the `slot[Z].status.questionable` register. Bit B3 decimal value: 8
B4 to B6	Not used.

Bit	Value
B7	`slot[Z].status.OSB` Set bit is a summary of the `slot[Z].status.operation` register. Bit B7 decimal value: 128
B8 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B0 and B7, set *measurementRegister* to 129 (which is the sum of 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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
slot[1].status.enable = slot[1].status.MSB
```

Sets the MSB bit of the status summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.*

This attribute contains the Measurement Event register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6,539 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.condition
measurementRegister = slot[Z].status.measurement.enable
measurementRegister = slot[Z].status.measurement.event
measurementRegister = slot[Z].status.measurement.ntr
measurementRegister = slot[Z].status.measurement.ptr
slot[Z].status.measurement.enable = measurementRegister
slot[Z].status.measurement.ntr = measurementRegister
slot[Z].status.measurement.ptr = measurementRegister
```

measurementRegister	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes read or write the Measurement Event registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Reserved for future use. Bit B0 decimal value: 1

Bit	Value
B1	`slot[Z].status.measurement.ILMT` Set bit is a summary of the `slot[Z].status.measurement.current_limit` register. Bit B1 decimal value: 2
B2	Not used.
B3	`slot[Z].status.measurement.PROT` Set bit indicates that current or voltage protection was tripped. Bit B3 decimal value: 8
B4 to B6	Not used.
B7	`slot[Z].status.measurement.ROF` Set bit is a summary of the `slot[Z].status.measurement.READING_OVERFLOW` register. Bit B7 decimal value: 128
B8	Reserved for future use. Bit B8 decimal value: 256
B9 to B10	Not used
B11	`slot[Z].status.measurement.INT` Set bit indicates that interlock has been asserted. Bit B11 decimal value: 2,048
B12	`slot[Z].status.measurement.INST` Set bit indicates that a bit in the measurement instrument summary register is set. Bit B12 decimal value: 4,096
B13 to B15	Not used

As an example, to set bit B7 of the measurement event enable register for slot 1, set `slot[1].status.measurement.enable = slot[1].status.measurement.ROF`.

In addition to the above constants, *`measurementRegister`* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *`measurementRegister`* to the sum of their decimal weights. For example, to set bits B1 and B7, set *`measurementRegister`* to 130 (which is the sum of 2 + 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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**

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	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example

```
slot[1].status.measurement.enable = slot[1].status.measurement.ROF
```

For slot 1, sets B7, the ROF bit of the Measurement Event enable register.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.current_limit.*

This attribute contains the measurement event current limit summary registers.

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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.current_limit.condition
measurementRegister = slot[Z].status.measurement.current_limit.enable
measurementRegister = slot[Z].status.measurement.current_limit.event
measurementRegister = slot[Z].status.measurement.current_limit.ntr
measurementRegister = slot[Z].status.measurement.current_limit.ptr
slot[Z].status.measurement.current_limit.enable = measurementRegister
slot[Z].status.measurement.current_limit.ntr = measurementRegister
slot[Z].status.measurement.current_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement event current limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the measurement event current limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 and bit B2 are set.

Bits B1 and B2 report the last status of the source while the output was turned on.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.current_limit.PSU1` Set bit indicates that the PSU 1 current limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.measurement.current_limit.PSU2` Set bit indicates that the PSU 2 current limit was exceeded. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[2].status.measurement.current_limit.enable = slot[2].status.measurement.curre
nt_limit.PSU2
```

For slot 2, sets the PSU channel 1 bit of the Measurement Event Current Limit Summary enable register.

Also see

Measurement event registers (on page 661)
slot[Z].status.measurement.instrument.psu[X].* (on page 418)

slot[Z].status.measurement.instrument.*

This attribute contains the registers of the measurement event instrument summary 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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.instrument.condition
measurementRegister = slot[Z].status.measurement.instrument.enable
measurementRegister = slot[Z].status.measurement.instrument.event
measurementRegister = slot[Z].status.measurement.instrument.ntr
measurementRegister = slot[Z].status.measurement.instrument.ptr
slot[Z].status.measurement.instrument.enable = measurementRegister
slot[Z].status.measurement.instrument.ntr = measurementRegister
slot[Z].status.measurement.instrument.ptr = measurementRegister
```

measurementRegister	The status of the measurement event instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Measurement Event Instrument Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume the value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 (PSU1) and bit B2 (PSU2) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.instrument.PSU1` Set bit indicates one or more enabled bits of the measurement event PSU 1 summary register is set. Bit B1 decimal value: 2 Binary value: 0000 0010

Bit	Value
B2	`slot[Z].status.measurement.instrument.PSU2` Set bit indicates one or more enabled bits of the measurement event PSU 2 summary register is set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.measurement.instrument.enable = slot[1].status.measurement.instrument.PSU1
```

Sets the PSU 1 bit of the measurement event instrument summary enable register for the instrument in slot 1 using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.instrument.psu[X].*

This attribute contains the registers of the measurement event PSU 1 summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	395 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.instrument.psu[X].condition
measurementRegister = slot[Z].status.measurement.instrument.psu[X].enable
measurementRegister = slot[Z].status.measurement.instrument.psu[X].event
measurementRegister = slot[Z].status.measurement.instrument.psu[X].ntr
measurementRegister = slot[Z].status.measurement.instrument.psu[X].ptr
slot[Z].status.measurement.instrument.psu[X].enable = measurementRegister
slot[Z].status.measurement.instrument.psu[X]= measurementRegister
slot[Z].status.measurement.instrument.psu[X].ptr = measurementRegister
```

measurementRegister	The status of the instrument measurement status PSU *Z* summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
Z	Module slot number
X	Channel of the PSU: 1 or 2

Details

These attributes are used to read or write to the measurement event PSU summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value `257` is returned for the enable register. The binary equivalent is `0000 0001 0000 0001`. This value indicates that bit B0 (VLMT) and bit B8 (BAV) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Reserved for future use. Bit B0 decimal value: 1
B1	`slot[Z].status.measurement.instrument.psu[X].ILMT` Set bit indicates that the current limit was exceeded. Bit B1 decimal value: 2
B2	Not used
B3	`slot[Z].status.measurement.instrument.psu[X].PROT` Set bit indicates that voltage or current protection was tripped. Bit B3 decimal value: 8
B4 to B6	Not used
B7	`slot[Z].status.measurement.instrument.psu[X].ROF` Set bit indicates that an overflow reading has been detected. Bit B7 decimal value: 128
B8	Reserved for future use. Bit B8 decimal value: 256
B9 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B7, set *measurementRegister* to 130 (which is the sum of 2 + 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

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
slot[1].status.measurement.instrument.psu[1].enable = slot[1].status.measurement.instrument.psu[1].ILMT
```

Sets the VLMT bit, B0, of the Measurement Event Summary enable register for PSU channel 1 in slot 1 using a constant.

Also see

Measurement event registers (on page 661)
slot[Z].psu[X].source.limiti (on page 393)
slot[Z].psu[X].source.protect.trippedi (on page 399)
slot[Z].psu[X].source.protect.trippedv (on page 400)

slot[Z].status.measurement.protection.*

This attribute contains the registers of the measurement event protection summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.protection.condition
measurementRegister = slot[Z].status.measurement.protection.enable
measurementRegister = slot[Z].status.measurement.protection.event
measurementRegister = slot[Z].status.measurement.protection.ntr
measurementRegister = slot[Z].status.measurement.protection.ptr
slot[Z].status.measurement.protection.enable = measurementRegister
slot[Z].status.measurement.protection.ntr = measurementRegister
slot[Z].status.measurement.protection.ptr = measurementRegister
```

measurementRegister	The status of the measurement event protection summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the measurement event protection summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.protection.PSU1` Set bit indicates the PSU 1 protection circuit was tripped. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.measurement.protection.PSU2` Set bit indicates the PSU 2 protection circuit was tripped. Bit B2 decimal value: 4 Binary value: 0000 0100

Bit	Value
B2 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal value of the bit to set.

Example 1

```
slot[1].status.measurement.protection.enable = slot[1].status.measurement.protecti
on.PSU1
```

Uses the constant to set bit 1 for slot 1, which is the PSU 1 bit of the Measurement Event Protection Summary enable register.

Example 2

```
slot[1].status.measurement.protection.enable = 2
```

Uses the decimal value to set bit 1 for slot 1, which is the PSU 1 bit of the Measurement Event Protection Summary enable register.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.reading_overflow.*

This attribute contains the measurement event reading overflow summary 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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.reading_overflow.condition
measurementRegister = slot[Z].status.measurement.reading_overflow.enable
measurementRegister = slot[Z].status.measurement.reading_overflow.event
measurementRegister = slot[Z].status.measurement.reading_overflow.ntr
measurementRegister = slot[Z].status.measurement.reading_overflow.ptr
slot[Z].status.measurement.reading_overflow.enable = measurementRegister
slot[Z].status.measurement.reading_overflow.ntr = measurementRegister
slot[Z].status.measurement.reading_overflow.ptr = measurementRegister
```

measurementRegister	The status of the measurement reading overflow summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the measurement event reading overflow summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 is set.

Bits B1 and B2 report the last measurement that was made. If a reading overflow occurs, the bit is set. The bit is cleared when a non-overflowed reading is made.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.reading_overflow.PSU1` Set bit indicates that an overflow reading has been detected for PSU 1. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.measurement.reading_overflow.PSU2` Set bit indicates that an overflow reading has been detected for PSU 2. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to constants, *measurementRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.measurement.reading_overflow.enable = slot[1].status.measurement.reading_overflow.PSU1
```

Sets the slot 1 PSU 1 bit of the Measurement Reading Overflow Summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.operation.*

These attributes manage the Operation Status Register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	8,217 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.condition
operationRegister = slot[Z].status.operation.enable
operationRegister = slot[Z].status.operation.event
operationRegister = slot[Z].status.operation.ntr
operationRegister = slot[Z].status.operation.ptr
slot[Z].status.operation.enable = operationRegister
slot[Z].status.operation.ntr = operationRegister
slot[Z]status.operation.ptr = operationRegister
```

operationRegister	The status of the operation status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate specific bit settings
Z	Module slot number

Details

These attributes read or write the Operation Status Registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `24` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 1000. This value indicates that bit B3 (`SWE`) and bit B4 (`MEAS`) 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	0	0	0	1	1	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	`slot[Z].status.operation.CAL` Set bit indicates that the summary bit of the `status.operation.calibrating` register has been set. Bit B0 decimal value: 1

Bit	Value
B1 to B2	Not used
B3	`slot[Z].status.operation.SWE` Set bit indicates that the summary bit from the `status.operation.sweeping` register is set. Bit B3 decimal value: 8
B4	`slot[Z].status.operation.MEAS` Set bit indicates that the summary bit of the `status.operation.measuring` register is set. Bit B4 decimal value: 16
B5 to B12	Not used
B13	`slot[Z].status.operation.INST` Set bit indicates that the summary bit from the `status.operation.instrument` register is set. Bit B13 decimal value: 8,192
B14 to B15	Not used

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = slot[1].status.operation.SWE + slot[1].status.operation.INST
slot[1].status.operation.enable = operationRegister
```

Uses constants to set the SWE and INST bits of the Operation Status Enable Register.

Example 2

```
-- decimal 24 = binary 0000 0000 0001 1000
operationRegister = 24
slot[1].status.operation.enable = operationRegister
```

Uses a decimal value to set the SWE and MEAS bits of the Operation Status Enable Register.

Also see

Operation Status Registers (on page 658)

slot[Z].status.operation.calibrating.*

This attribute contains the operation status calibration summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.calibrating.condition
operationRegister = slot[Z].status.operation.calibrating.enable
operationRegister = slot[Z].status.operation.calibrating.event
operationRegister = slot[Z].status.operation.calibrating.ntr
operationRegister = slot[Z].status.operation.calibrating.ptr
slot[Z].status.operation.calibrating.enable = operationRegister
slot[Z].status.operation.calibrating.ntr = operationRegister
slot[Z].status.operation.calibrating.ptr = operationRegister
```

operationRegister	The status of the operation calibrating event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the operation status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.operation.calibrating.PSU1` Set bit indicates that PSU1 is unlocked for calibration. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.operation.calibrating.PSU2` Set bit indicates that PSU2 is unlocked for calibration. Bit B2 decimal value: 4 Binary value: 0000 0100

Bit	Value
B3 to B15	Not used

Example

```
slot[1].status.operation.calibrating.enable = slot[1].status.operation.calibrating
.PSU1
```

Sets the PSU1 bit of the operation status calibration summary enable register for slot 1 using a constant.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.instrument.*

This attribute contains the operation status instrument summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.instrument.condition
operationRegister = slot[Z].status.operation.instrument.enable
operationRegister = slot[Z].status.operation.instrument.event
operationRegister = slot[Z].status.operation.instrument.ntr
operationRegister = slot[Z].status.operation.instrument.ptr
slot[Z].status.operation.instrument.enable = operationRegister
slot[Z].status.operation.instrument.ntr = operationRegister
slot[Z].status.operation.instrument.ptr = operationRegister
```

operationRegister	The status of the operation event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the operation status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used

B1	`slot[Z].status.operation.instrument.PSU1` Set bit indicates one or more enabled bits for the operation status PSU 1 summary register are set. Bit B1 decimal value: 2
B2	`slot[Z].status.operation.instrument.PSU2` Set bit indicates one or more enabled bits for the operation status PSU 2 summary register are set. Bit B2 decimal value: 4
B3 to B15	Not used

As an example, to set bit B1 of the operation status instrument summary enable register, set `slot[1].status.operation.instrument.enable = slot[1].status.operation.instrument.PSU1`

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B2, set *`operationRegister`* to 6 (the sum of 2 + 4).

Bit	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = slot[1].status.operation.instrument.PSU1 +
   slot[1].status.operation.instrument.PSU2
slot[1].status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B2 of the Operation Status Instrument Summary enable register using constants.

Example 2

```
-- 6 = binary 0000 0000 0000 0110
operationRegister = 6
slot[1].status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B2 of the Operation Status Instrument Summary enable register using a decimal value.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.instrument.psu[X].*

This attribute contains the operation status PSU summary 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	25 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.instrument.psu[X].condition
operationRegister = slot[Z].status.operation.instrument.psu[X].enable
operationRegister = slot[Z].status.operation.instrument.psu[X].event
operationRegister = slot[Z].status.operation.instrument.psu[X].ntr
operationRegister = slot[Z].status.operation.instrument.psu[X].ptr
slot[Z].status.operation.instrument.psu[X].enable = operationRegister
slot[Z].status.operation.instrument.psu[X].ntr = operationRegister
slot[Z].status.operation.instrument.psu[X].ptr = operationRegister
```

operationRegister	The status of the operation status PSU summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number
X	Power supply unit (PSU) channel (for example `slot[1].status.operation.instrument.psu[2].enable` applies to PSU channel 1 in slot 2)

Details

These attributes are used to read or write to the operation status PSU[*X*] summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `24` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 1000. This value indicates that bit B0 and bit B10 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	0	0	0	1	1	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	`slot[Z].status.operation.instrument.psu[X].CAL` Set bit indicates that PSU is unlocked for calibration. Bit B0 decimal value: 1

Bit	Value and description
B1 to B2	Not used
B3	`slot[Z].status.operation.instrument.psu[X].SWE` Set bit indicates that PSU is sweeping. Bit B3 decimal value: 8
B4	`slot[Z].status.operation.instrument.psu[X].MEAS` Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B4 decimal value: 16
B5 to B15	Not used

As an example, to set bit B0 of the operation status PSU 1 summary enable register, set `slot[1].status.operation.instrument.psu[1].enable = slot[1].status.operation.instrument.psu[1].CAL`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B0 and B4, set *`operationRegister`* to 17 (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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
slot[1].status.operation.instrument.psu[1].enable = slot[1].status.operation.instrument.psu[1].SWE
```

Use a constant to set bit B3 of the operation status PSU 1 summary enable register for slot 1.

Example 2

```
slot[1].status.operation.instrument.psu[1].enable = 24
```

Use the decimal value to set bits B3 and B4 of the operation status PSU 1 summary enable register for slot 1.

Also see

Operation Status Registers (on page 658)

slot[Z].status.operation.measuring.*

This attribute contains the operation status measuring summary 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	Not available	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.measuring.condition
operationRegister = slot[Z].status.operation.measuring.enable
operationRegister = slot[Z].status.operation.measuring.event
operationRegister = slot[Z].status.operation.measuring.ntr
operationRegister = slot[Z].status.operation.measuring.ptr
slot[Z].status.operation.measuring.enable = operationRegister
slot[Z].status.operation.measuring.ntr = operationRegister
slot[Z].status.operation.measuring.ptr = operationRegister
```

operationRegister	The status of the operation status measuring summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the operation status measuring summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.operation.measuring.PSU1` Bit is set when PSU 1 is making an overlapped measurement. It is not set when making a normal synchronous measurement. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.operation.measuring.PSU2` Bit is set when PSU 2 is making an overlapped measurement. It is not set when making a normal synchronous measurement. Bit B2 decimal value: 4 Binary value: 0000 0100

Bit	Value
B3 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *operationRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.operation.measuring.enable = slot[1].status.operation.measuring.PSU
1
```

Uses a constant to set the PSU1 bit of the operation status measuring summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.sweeping.*

This attribute contains the operation status sweeping summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.sweeping.condition
operationRegister = slot[Z].status.operation.sweeping.enable
operationRegister = slot[Z].status.operation.sweeping.event
operationRegister = slot[Z].status.operation.sweeping.ntr
operationRegister = slot[Z].status.operation.sweeping.ptr
slot[Z].status.operation.sweeping.enable = operationRegister
slot[Z].status.operation.sweeping.ntr = operationRegister
slot[Z].status.operation.sweeping.ptr = operationRegister
```

operationRegister	The status of the operation status sweeping summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the operation status sweeping summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.operation.sweeping.PSU1` Set bit indicates that PSU 1 has a source action trigger block in the trigger model when the trigger model is initiated. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.operation.sweeping.PSU2` Set bit indicates PSU 2 has a source action trigger block in the trigger model when the trigger model is initiated.. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *operationRegister* to 6 (the sum of 2 + 4).

Example

```
slot[1].psu[1].source.levelv = 2
slot[1].psu[1].trigger.source.linearv(0,5,6)
slot[1].psu[1].source.output = 1
TM = slot[1].trigger.model
TM.delete("SweepTriggerModel")
TM.create("SweepTriggerModel")
TM.addblock.source.action.step("SweepTriggerModel", "sourceActionStep", 1)
TM.addblock.delay.constant("SweepTriggerModel", "delay50ms", 0.05)
TM.addblock.branch.counter("SweepTriggerModel", "branchCounter", "sourceActionStep
", 6)
TM.initiate("SweepTriggerModel")
print(slot[1].status.operation.sweeping.condition)
waitcomplete()
print("Done")
```

Set source for a level of 2 V.

Set up a voltage linear sweep.

Turn on the source output.

Set up the trigger model `SweepTriggerModel` to run the sweep.

Run the trigger model.

Return the condition of the sweep.

Wait for the results.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.questionable.*

These attributes manage the questionable status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	13,056 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.condition
questionableRegister = slot[Z].status.questionable.enable
questionableRegister = slot[Z].status.questionable.event
questionableRegister = slot[Z].status.questionable.ntr
questionableRegister = slot[Z].status.questionable.ptr
slot[Z].status.questionable.enable = questionableRegister
slot[Z].status.questionable.ntr = questionableRegister
slot[Z].status.questionable.ptr = questionableRegister
```

questionableRegister	The status of the questionable status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the questionable status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, if a value of 4,352 is read as the value of the condition register, the binary equivalent is 0001 0001 0000 0000. This value indicates that bits B8 and B12 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B7	Not used	Not available

Bit	Value	Description
B8	`slot[Z].status.questionable.CAL`	An enabled bit in the questionable status calibration summary event register is set. Bit B8 decimal value: 256
B9	Reserved for future	Bit B9 decimal value: 512
B10 to B11	Not used	Not available
B12	`slot[Z].status.questionable.OTEMP`	An enabled bit in the questionable status overtemperature summary event register is set. Bit B12 decimal value: 4,096
B13	`slot[Z].status.questionable.INST`	An enabled bit in the questionable status instrument summary event register is set. Bit B13 decimal value: 8,192
B14 to B15	Not used	Not available

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights. For example, to set bits B12 and B13, set *`questionableRegister`* to 12,288 (which is the sum of 4,096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
slot[1].status.questionable.enable = slot[1].status.questionable.OTEMP
```

Uses a constant to set the OTEMP bit of the questionable status enable register for slot 1.

Example 2

```
slot[1].status.questionable.enable = slot[1].status.questionable.CAL
slot[1].status.questionable.enable = 256
```

Both of these commands set the CAL enable bit (B8).

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)

slot[Z].status.questionable.calibration.*

This attribute contains the questionable status calibration summary 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	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.calibration.condition
questionableRegister = slot[Z].status.questionable.calibration.enable
questionableRegister = slot[Z].status.questionable.calibration.event
questionableRegister = slot[Z].status.questionable.calibration.ntr
questionableRegister = slot[Z].status.questionable.calibration.ptr
slot[Z].status.questionable.calibration.enable = questionableRegister
slot[Z].status.questionable.calibration.ntr = questionableRegister
slot[Z].status.questionable.calibration.ptr = questionableRegister
```

questionableRegister	The status of the questionable status calibration summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits that are set
Z	Module slot number

Details

These attributes are used to read or write to the questionable status calibration summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used
B1	`slot[Z].status.questionable.calibration.PSU1` Set bit indicates that the PSU 1 calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B1 decimal value: 2 Binary value: 0000 0010

Bit	Value and description
B2	`slot[Z].status.questionable.calibration.PSU2` Set bit indicates that the PSU 2 calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *questionableRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.questionable.calibration.enable = slot[1].status.questionable.calibration.PSU1
```

Uses a constant to set the PSU1 bit of the questionable status calibration summary enable register for slot 1.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.questionable.* (on page 434)

slot[Z].status.questionable.instrument.*

This attribute contains the questionable status instrument summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.instrument.condition
questionableRegister = slot[Z].status.questionable.instrument.enable
questionableRegister = slot[Z].status.questionable.instrument.event
questionableRegister = slot[Z].status.questionable.instrument.ntr
questionableRegister = slot[Z].status.questionable.instrument.ptr
slot[Z].status.questionable.instrument.enable = questionableRegister
slot[Z].status.questionable.instrument.ntr = questionableRegister
slot[Z].status.questionable.instrument.ptr = questionableRegister
```

questionableRegister	The status of the questionable status instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Questionable Status Instrument Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.questionable.instrument.PSU1` Set bit indicates one or more enabled bits for the PSU channel 1 Questionable register are set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.questionable.instrument.PSU2` Set bit indicates one or more enabled bits for the PSU channel 2 Questionable register are set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.questionable.instrument.enable = slot[1].status.questionable.instrument.PSU1
```

Uses a constant to set the PSU channel 1 bit of the questionable status instrument summary enable register for slot 1.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.questionable.* (on page 434)

slot[Z].status.questionable.instrument.psu[X].*

This attribute contains the questionable status PSU summary 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	Not applicable	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	4,864 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.instrument.psu[X].condition
questionableRegister = slot[Z].status.questionable.instrument.psu[X].enable
questionableRegister = slot[Z].status.questionable.instrument.psu[X].event
questionableRegister = slot[Z].status.questionable.instrument.psu[X].ntr
questionableRegister = slot[Z].status.questionable.instrument.psu[X].ptr
slot[Z].status.questionable.instrument.psu[X].enable = questionableRegister
slot[Z].status.questionable.instrument.psu[X].ntr = questionableRegister
slot[Z].status.questionable.instrument.psu[X].ptr = questionableRegister
```

questionableRegister	The status of the questionable status PSU summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits that are set
Z	Module slot number
X	Channel of the PSU: 1 or 2

Details

These attributes are used to read or write to the questionable status instrument PSU summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `7.68000e+02` (which is 768) is read as the value of the condition register, the binary equivalent is 0000 0011 0000 0000. This value indicates that bit B8 and bit B12 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B7	Not used.

Bit	Value and description
B8	`slot[Z].status.questionable.instrument.psu[X].CAL` Set bit indicates that the calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B8 decimal value: 256
B9	`slot[Z].status.questionable.instrument.psu[X].WARMUP` Set bit indicates that the channel is warmed up and ready for autocalibration. Bit B9 decimal value: 512
B10 to B11	Not used.
B12	Reserved for future use. Bit B12 decimal value: 4,096
B13 to B15	Not used

As an example, to set bit B8 of the questionable status PSU channel 1 summary enable register for slot 1, set `slot[1].status.questionable.instrument.psu[1].enable = slot[1].status.questionable.instrument.psu[1].CAL`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
questionableRegister = slot[1].status.questionable.instrument.psu[1].CAL
slot[1].status.questionable.instrument.psu[1].enable = questionableRegister
```

Uses constants to set bit B8 of the questionable status PSU summary enable register for slot 1.

Also see

Questionable Status Registers (on page 673)
status.operation.* (on page 244)

slot[Z].status.questionable.over_temperature.*

his attribute contains the Questionable Status Overtemperature Summary 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	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.over_temperature.condition
questionableRegister = slot[Z].status.questionable.over_temperature.enable
questionableRegister = slot[Z].status.questionable.over_temperature.event
questionableRegister = slot[Z].status.questionable.over_temperature.ntr
questionableRegister = slot[Z].status.questionable.over_temperature.ptr
slot[Z].status.questionable.over_temperature.enable = questionableRegister
slot[Z].status.questionable.over_temperature.ntr = questionableRegister
slot[Z].status.questionable.over_temperature.ptr = questionableRegister
```

questionableRegister	The status of the questionable status overtemperature summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the questionable status overtemperature summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.questionable.over_temperature.PSU1` Set bit indicates that an overtemperature condition was detected on PSU 1. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.questionable.over_temperature.PSU2` Set bit indicates that an overtemperature condition was detected on PSU 2. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

As an example, to set bit B1 of the questionable status overtemperature summary enable register for slot 1, set `slot1.status.questionable.instrument.enable = status.questionable.instrument.PSU1`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B2, set *`questionableRegister`* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.questionable.over_temperature.enable = slot[1].status.questionable.
over_temperature.PSU1
```

Uses a constant to set the PSU 1 bit in the questionable status overtemperature summary enable register for slot 1.

Also see

Questionable Status Register (on page 673)
slot[Z].status.questionable.* (on page 434)

SMU module TSP commands

Introduction

The TSP commands available for the MP5000 Series source-measure unit (SMU) modules are listed in alphabetical order.

bufferVar.appendmode

This attribute sets the state of the append mode of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0 (off)

Usage

```
state = bufferVar.appendmode
bufferVar.appendmode = state
```

state	The reading buffer append mode; set to one of the following: ▪ Append mode off; clear the buffer and then add new data to buffer: `0` ▪ Append mode on; new data is added to the existing buffer content: `1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

Assign a value to this attribute to enable or disable the buffer append mode. This value can only be changed when the buffer is empty. Use *bufferVar*`.clear()` to empty the buffer.

If the append mode is set to `0` (off), any previously stored readings in the buffer are cleared before new ones are stored. If append mode is set to `1`, any previously stored readings remain in the buffer and new readings are added to the buffer following the stored readings.

With append mode set to `1` (on), the first new measurement is stored at reading buffer location `[`*n*`+1]`, where *n* is the number of readings stored in the buffer. When the fill mode is set to once, measurements are only appended to a buffer if the measure count is less than the available space in the buffer. If the buffer cannot hold the readings, an error is returned and no measurements are made.

Example 1

```
buffer1 = slot[1].smu[1].makebuffer(100)
buffer1.clear()
buffer1.appendmode = 1
print(buffer1.appendmode)
```

Append new readings to contents of the reading buffer `buffer1` created for channel 1 of the module installed in slot 1.

Example output:

```
1.00000e+00
```

Example 2

```
slot[2].smu[2].defbuffer1.clear()
slot[2].smu[2].defbuffer1.appendmode = 1
slot[2].smu[2].measure.v(slot[2].smu[2].defbuffer1)
printbuffer(1, slot[2].smu[2].defbuffer1.n, slot[2].smu[2].defbuffer1)
slot[2].smu[2].measure.v(slot[2].smu[2].defbuffer1)
printbuffer(1, slot[2].smu[2].defbuffer1.n, slot[2].smu[2].defbuffer1)
```

Clear `defbuffer1` on slot 2, SMU channel 2.

Set the append mode to enable.

Make a measurement and store the data in `defbuffer1`.

Return the data from `defbuffer1`.

Make another measurement.

Return the data from `defbuffer1`.

Example output:

```
1.91e-05
1.91e-05, 1.91e-05
```

Also see

bufferVar.clear (on page 447)

bufferVar.basetimestampfractional

This attribute contains the fractional seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

fractionalSec = *bufferVar*.basetimestampfractional

fractionalSec	The fractional seconds portion of the timestamp when the first reading was stored
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The read-only base timestamp attributes contain the timestamp of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 AM January 1, 1970 (UTC) that the measurement was performed and stored.

The base timestamp of the reading buffer is divided into two parts. This attribute contains the fractional portion of the base timestamp (with 100 ns of precision) and can be used with the seconds portion of the base timestamp.

Example

```
buffer = slot[3].smu[2].defbuffer2
print(tostring(buffer.basetimestampseconds))
```

Return the timestamp for the first reading stored in defbuffer2 on channel 2 of the SMU in slot 3.

Example output:

```
1756222247
```

This output indicates that the timestamp is 1,756,222,247 seconds or Tuesday, August 26, 2025, at 11:30:47 AM.

Also see

bufferVar.basetimestampseconds (on page 445)
Reading buffers (on page 96)

bufferVar.basetimestampseconds

This attribute contains the whole seconds portion of the timestamp, which indicates when the first reading was stored in the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

nonFracSeconds = *bufferVar*.basetimestampseconds

nonFracSeconds	The nonfractional seconds portion of the timestamp when the first reading was stored
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The read-only base timestamp attributes contain the timestamp of the first reading stored in a buffer (`rb[1]` stored in reading buffer `rb`). The timestamp is the number of seconds since 12:00 AM January 1, 1970 (UTC) that the measurement was performed and stored.

The base timestamp of the reading buffer is divided into two parts. This attribute contains the whole seconds portion of the base timestamp (resolution of 1 second.) It can be used with the fractional seconds portion of the base timestamp for additional precision.

Example

```
reset()
buffer1 = slot[2].smu[1].makebuffer(5)
buffer1.clear()
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.v(buffer1)
printbuffer(1, buffer1.n, buffer1.readings)
print(buffer1.basetimestampseconds .. buffer1.basetimestampfractional)
```

Create the reading buffer `buffer1`.

Clear the buffer.

Set the measurement count to 3.

Set up voltage measurements to go into buffer1.

Return the readings from buffer1.

Return the seconds and fractional seconds.

Example output:

```
-1.045526296e-04, -1.023707810e-04, -1.035681344e-04
127210.8870559
```

Also see

bufferVar.basetimestampfractional (on page 444)

bufferVar.cachemode

This attribute enables or disables the cache of the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	1 (enabled)

Usage

```
cacheMode = bufferVar.cachemode
bufferVar.cachemode = cacheMode
```

cacheMode	The reading buffer cache mode; set to one of the following: ▪ Cache mode disabled (off): `0` ▪ Cache mode enabled (on): `1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The reading buffer cache improves access speed to reading buffer data. However, if you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you initiate successive sweeps but do not reconfigure the sweep measurements. It can also happen when the *bufferVar*`.fillmode` attribute is set to `1`, which may cause reading buffer data to be overwritten.

To prevent the return of stale cache data, disable the cache or make an explicit call to *bufferVar*`.clearcache()`.

Example 1

```
buffer1 = slot[1].smu[1].makebuffer(100)
buffer1.cachemode = 0
```

Disables the reading buffer cache of user-defined reading buffer `buffer1`.

Example 2

```
slot[2].smu[2].defbuffer1.cachemode = 0
print(slot[2].smu[2].defbuffer1.cachemode)
```

Disables the reading buffer cache of default reading buffer `defbuffer1` on SMU channel 2 in slot 2.

Also see

bufferVar.clearcache() (on page 448)
bufferVar.fillmode (on page 450)

bufferVar.capacity

This attribute contains the capacity of the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

```
maxNumber = bufferVar.capacity
```

maxNumber	The maximum number of readings the buffer can store
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This read-only attribute returns the number of readings that can be stored in the buffer. The buffer capacity does not change as readings fill the buffer.

Example 1

```
buffer1 = slot[1].smu[1].makebuffer(100)
maxNumber = buffer1.capacity
print(maxNumber)
```

Returns the maximum number of readings that can be stored in `buffer1`:

```
1.00000e+02
```

Example 2

```
print(slot[2].smu[2].defbuffer1.capacity)
```

Returns the maximum number of readings that can be stored in the SMU channel 2 `defbuffer1` buffer on slot 2.

Also see

slot[Z].smu[X].makebuffer() (on page 493)

bufferVar.clear()

This function empties the reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
bufferVar.clear()
```

bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This function clears all readings and associated columns that are enabled from the specified buffer. It also clears the buffer cache.

Example 1

```
buffer1 = slot[1].smu[1].debuffer1
buffer1.clear()
```

Clears readings from the `debuffer1` buffer on channel 1 of the module in slot 1.

Example 2

```
slot[2].smu[2].defbuffer1.clear()
```

Clears readings from the SMU channel 2 buffer `defbuffer1` on slot 1.

Also see

bufferVar.clearcache() (on page 448)

bufferVar.clearcache()

This function clears the buffer cache of the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar`.clearcache()`

bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

If you run successive operations that overwrite reading buffer data, the reading buffer may return stale cache data. This can happen when you:

- Initiate successive sweeps without reconfiguring the sweep measurements. Watch for this when running Lua code remotely on more than one node, because values in the reading buffer cache may change while the Lua code is running.
- Overwrite data in the reading buffer by setting the *bufferVar*`.fillmode` attribute to `1`.

To avoid this, you can include explicit calls to the *bufferVar*`.clearcache()` function to remove stale values from the reading buffer cache.

Example 1

```
buffer1 = slot[1].smu[1].defbuffer1
buffer1.clearcache()
```

Clear all readings from the reading buffer cache of `defbuffer1` on channel 1 of the module in slot 1.

Example2

```
slot[2].smu[2].defbuffer1.clearcache()
```

Clear all readings from the reading buffer cache for `defbuffer1` of SMU channel 2 in slot 2..

Also see

bufferVar.fillmode (on page 450)
Reading buffers (on page 96)

bufferVar.fillcount

This attribute sets the reading buffer fill mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
fillCount = bufferVar.fillcount
bufferVar.fillcount = fillCount
```

fillCount	The reading buffer fill count; set to `0` to use the full capacity of the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

The reading buffer fill count sets the number of readings to store before restarting at index 1. If the value is `0`, the full capacity of the buffer is used. Use this attribute to control when the SMU restarts filling the buffer at index 1, rather than having it restart when the buffer becomes full.

If the *bufferVar*`.fillcount` attribute is set to a value higher than the capacity of the buffer, the behavior is the same as having a fill count of 0. When the buffer capacity is reached, storage continues at the starting index.

This attribute is only used if *bufferVar*`.fillmode` is set to 1.

Example 1

```
buffer1 = slot[1].smu[1].defbuffer1
buffer1.fillmode = 1
buffer1.fillcount = 50
```

Sets the fill count for `defbuffer1` on channel 1 of the module installed in slot 1 to 50.

Example 2

```
slot[1].smu[1].defbuffer1.fillcount = 50
```

Sets the fill count for the slot 1 SMU channel 1 reading buffer `defbuffer1` to 50.

Also see

bufferVar.fillmode (on page 450)

bufferVar.fillmode

This attribute sets the reading buffer fill mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	0

Usage

```
fillMode = bufferVar.fillmode
bufferVar.fillmode = fillMode
```

fillMode	The reading buffer fill mode; set to one of the following: ▪ Fill once (do not overwrite old data): `0` ▪ Fill continuously; new readings restart at index 1 after acquiring reading at index *bufferVar*`.fillcount`: `1`
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

When this attribute is set to fill once (`0`), the reading buffer does not overwrite readings. If the requested measurement operation would exceed the size of the buffer, no measurements are made and an error is returned.

When this attribute is set to fill continuously (`1`) and append mode is set on, new readings are added sequentially until the buffer holds *bufferVar*`.fillcount` elements. The next reading overwrites the reading at index 1, the following reading overwrites the reading at index 2, and so on.

Example 1

```
buffer1 = slot[1].smu[1].makebuffer(100)
buffer1.fillmode = 1
```

Enable window fill mode for `buffer1`. The readings will fill the buffer until the fillcount index and then wrap around.

Example 2

```
slot[1].smu[1].defbuffer1.fillmode = 0
```

Sets the reading mode of the `defbuffer1` for channel 1 of the module installed in slot 1 so that it does not overwrite readings.

Also see

bufferVar.appendmode (on page 443)
bufferVar.fillcount (on page 449)

bufferVar.measurefunctions

This attribute contains the measurement function that was used to acquire readings stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

measurefunction = *bufferVar*.measurefunctions[*N*]

measurefunction	The measurement function used (current, voltage, ohms, or watts) to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The measurefunctions buffer column is a Lua table array of strings that indicates the function that was measured for the reading.

NOTE: This command does not report reliable data for trigger models.

Example 1

```
buffer1 = slot[2].smu[1].makebuffer(100)
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.v(buffer1)
printbuffer(1, buffer1.n, buffer1.readings, buffer1.measurefunctions)
print(buffer1.readings[3])
```

Create a reading buffer named buffer1 on slot 1, SMU channel 1.

Set the measure count to 3.

Make voltage measurements and store them in buffer1.

Print the measurement and the measurement function for the readings in the reading buffer.

Print the third reading.

Output example:

```
-5.1e-06, Voltage, -5.1e-06, Voltage, -4.9e-06, Voltage
-4.9e-06
```

Example 2

```
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.v(slot[2].smu[1].defbuffer1)
printbuffer(1, slot[2].smu[1].defbuffer1.n, slot[2].smu[1].defbuffer1.readings, sl
ot[2].smu[1].defbuffer1.measurefunctions)
```

Set the measure count to 3.

Make voltage measurements and store them in defbuffer1.

Print the measurement and the measurement function for the readings in the reading buffer. Output example:

```
-5.1e-06, Voltage, -5.1e-06, Voltage, -4.9e-06, Voltage
```

Also see

None

bufferVar.measureranges

This attribute contains the measurement range values that were used for readings stored in a specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle bufferVar.clear()	Not applicable	Not applicable

Usage

measurerange = *bufferVar*.measureranges[*N*]

measurerange	The measurement range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to bufferVar.n

Details

The measureranges buffer column is a Lua table array of full-scale range values for the measure range that was used when the measurement was made.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[1].smu[1].makebuffer(100)
printbuffer(1, 5, buffer1.measureranges[1])
```

Prints the measure range for the first five readings in a reading buffer for channel 1 of the module installed in slot 1.

Also see

None

bufferVar.n

This attribute contains the number of readings in the buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

numberOfReadings = *bufferVar*.n

numberOfReadings	The number of readings stored in the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

This read-only attribute contains the number of readings stored in the buffer.

Example 1

```
buffer1 = slot[2].smu[1].makebuffer(100)
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.i(buffer1)
numberOfReadings = buffer1.n
print(numberOfReadings)
```

Create a reading buffer named `buffer1` on slot 2, SMU channel 1.

Set the measure count to 3.

Make current measurements and store the readings in `buffer1`.

Set the `numberOfReadings` variable to the number of readings stored in `buffer1`.

Return the number of readings:

```
3
```

Example 2

```
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.i(slot[2].smu[1].defbuffer1)
numberOfReadings = slot[2].smu[1].defbuffer1.n
print(numberOfReadings)
```

Set the measure count to 3.

Make current measurements and store the readings in `defbuffer1` on SMU channel 1 in slot 2.

Set the `numberOfReadings` variable to the number of readings stored in `defbuffer1.n`.

Return the number of readings:

```
3
```

Also see

None

bufferVar.readings

This attribute contains the acquired readings stored in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

reading = *bufferVar*.readings[*N*]

reading	The value of the reading in the specified reading buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: `1` to *bufferVar*.n

Details

The readings buffer column is a Lua table array of the readings stored in the reading buffer. This column holds the same data that is returned when the reading buffer is accessed directly; that is, *bufferVar*[2] and *bufferVar*.readings[2] access the same value.

Example 1

```
buffer1 = slot[1].smu[1].makebuffer(100)
printbuffer(1, buffer1.n, buffer1.readings)
```

Prints the first five readings in a reading buffer for channel 1 of the module installed in slot 1.

Example 2

```
slot[2].smu[1].measure.count = 3
slot[2].smu[1].measure.i(slot[2].smu[1].defbuffer1)
printbuffer(1, 3, slot[2].smu[1].defbuffer1.readings)
```

Set measure count to 3.

Make current measurements using SMU channel 1 in slot 2.

Return the first three readings from defbuffer1.

Also see

None

bufferVar.sourcefunctions

This attribute contains the source function that was used when the readings were stored in a specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcefunction = *bufferVar*.sourcefunctions[*N*]

sourcefunction	The source function used (Current or Voltage)
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The sourcefunctions buffer column is a Lua table array of strings that indicate the source function at the time of the measurement.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[2].smu[1].makebuffer(10)
buffer1.clear()
slot[2].smu[1].measure.count = 10
slot[2].smu[1].source.levelv = 2.2
slot[2].smu[1].source.output = 1
slot[2].smu[1].measure.v(buffer1)
slot[2].smu[1].source.output = 0
printbuffer(1, buffer1.n, buffer1.readings, buffer1.sourcefunctions, buffer1.sourceoutputstates)
```

Create the reading buffer `buffer1` on slot 2 SMU channel 1.

Clear the buffer.

Set the measure count to 20.

Set the source voltage level to 2.2.

Turn the output on.

Make measurements and store them in `buffer1`.

Turn the output off.

Show each reading with the source function and source output state.

Example output:

```
2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On
```

Also see

None

bufferVar.sourceoutputstates

This attribute indicates the state of the source output for readings stored in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

state = *bufferVar*.sourceoutputstates[*N*]

state	The output state when reading *N* of the specified buffer was acquired: On or Off
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The sourceoutputstates buffer column is a Lua table array of strings that indicate the state of the source output (off or on) at the time of the measurement.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[2].smu[1].makebuffer(10)
buffer1.clear()
slot[2].smu[1].measure.count = 10
slot[2].smu[1].source.levelv = 2.2
slot[2].smu[1].source.output = 1
slot[2].smu[1].measure.v(buffer1)
slot[2].smu[1].source.output = 0
printbuffer(1, buffer1.n, buffer1.readings, buffer1.sourcefunctions, buffer1.sourceoutputstates)
```

Create the reading buffer buffer1 on slot 2 SMU channel 1.

Clear the buffer.

Set the measure count to 20.

Set the source voltage level to 2.2.

Turn the output on.

Make measurements and store them in buffer1.

Turn the output off.

Show each reading with the source function and source output state.

Example output:

```
2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19993e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On, 2.19994e+00, Voltage, On
```

Also see

None

bufferVar.sourceranges

This attribute contains the source range used for readings stored in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcerange = *bufferVar*.sourceranges[*N*]

sourcerange	The source range used to acquire reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The sourceranges buffer recall attribute is a Lua table array of full-scale range values for the source range used when each measurement was made.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[2].smu[1].makebuffer(20)
buffer1.clear()
slot[2].smu[1].measure.count = 20
slot[2].smu[1].measure.v(buffer1)
printbuffer(1, 5, buffer1.readings, buffer1.sourceranges)
```

Create reading buffer buffer1 that holds 20 readings.

Clear the buffer.

Set the measure count to 20.

Make voltage measurements and store them in `buffer1`.

Print the readings and source ranges.

Example output:

```
2.19992e+00, 6.00000e+00, 2.19992e+00, 6.00000e+00, 2.19992e+00, 6.00000e+00, 2.19
992e+00, 6.00000e+00, 2.19992e+00, 6.00000e+00
```

Also see

bufferVar.n (on page 452)

bufferVar.sourcevalues

This attribute contains the source levels that were programmed when the readings in the reading buffer were acquired.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

sourcevalue = *bufferVar*.sourcevalues[*N*]

sourcevalue	The source value programmed while acquiring reading number *N* in the specified buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

The sourcevalues buffer column is a Lua table of the programmed sourced value at the time of the reading.

NOTE: This command does not report reliable data for trigger models.

Example

```
buffer1 = slot[2].smu[1].makebuffer(20)
buffer1.clear()
slot[2].smu[1].measure.count = 20
slot[2].smu[1].source.levelv = 2.2
slot[2].smu[1].source.output = 1
slot[2].smu[1].measure.v(buffer1)
slot[2].smu[1].source.output = 0
printbuffer(1, 5, buffer1.readings, buffer1.sourceranges, buffer1.sourcevalues)
```

Create reading buffer buffer1 on SMU channel 1 in slot 2.

Clear the buffer.

Set the measurement count to 20.

Set the source voltage level to 2.2.

Turn the output on.

Request voltage measurements to be stored in buffer1

Turn the output off..

Return the reading, source range, and source value for the first five measurements.

Example output:

```
2.19923e+00, 6.00000e+00, 2.20000e+00, 2.19992e+00, 6.00000e+00, 2.20000e+00, 2.19
991e+00, 6.00000e+00, 2.20000e+00, 2.19991e+00, 6.00000e+00, 2.20000e+00, 2.19991e
+00, 6.00000e+00, 2.20000e+00
```

Also see

None

bufferVar.statuses

This attribute contains the status values of readings in the specified reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

```
statusInformation = bufferVar.statuses[N]
```

statusInformation	The status value when reading *N* of the specified buffer was acquired
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: `1` to *bufferVar.n*

Details

This buffer column is a Lua table of the status values for each reading in the buffer. The status values are floating-point numbers that encode the status value.

The buffer status values are not updated during trigger model operation.

The buffer status bits are listed in the following table.

Bit	Name	Decimal value	Hex value	Description
B0	Reserved	1	0x01	Reserved for future use
B1	Overtemp	2	0x02	Overtemperature shutdown; measurement performed with output off
B2	AutoRangeMeas	4	0x04	Measurement autorange enabled for this reading
B3	AutoRangeSrc	8	0x08	Source autorange enabled for this reading
B4	4Wire	16	0x10	4-wire remote sense or autosense mode enabled for this reading
B5	Rel	32	0x20	Relative offset applied to reading
B6	Limit	64	0x40	The output level was reduced by the source limit for this reading
B7	Reserved	128	0x80	Reserved for future use

Example

```
buffer1 = slot[1].smu[1].makebuffer(100)
printbuffer(1, 5, buffer1.statuses)
```

Prints the status values for the first five readings in a reading buffer for channel 1 of the module installed in slot 1.

Also see

None

bufferVar.timestampresolution

This attribute contains the resolution of the timestamp for each reading stored in a reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle	Not applicable	100e-9 (100 ns)

Usage

```
resolution = bufferVar.timestampresolution
bufferVar.timestampresolution = resolution
```

resolution	Timestamp resolution in seconds (minimum 100 ns; rounded to an even power of 200 ns)
bufferVar	The reading buffer; can be a user-defined or default reading buffer

Details

Assigning a value to this attribute sets the resolution for the timestamps. Reading this attribute returns the timestamp resolution value.

This value can only be changed when the buffer is empty. Empty the buffer using the *bufferVar.clear()* function.

The finest timestamp resolution is 100e−9 seconds (100 ns). At this resolution, the reading buffer can store unique timestamps for up to 7 minutes. You can increase this value for long tests.

The value specified when setting this attribute is rounded to an even power of 200 ns.

This setting does not affect the resolution of the base timestamp of the reading buffer.

Example

```
buffer1 = slot[1].smu[1].makebuffer(100)
buffer1.timestampresolution = 8e-6
print(buffer1.timestampresolution)
```

Creates a new reading buffer and sets its timestamp resolution to 8 µs. Due to rounding, the actual resolution is set to 12.8 µs.

Example output:

```
1.28000e-05
```

Also see

bufferVar.clear() (on page 447)
bufferVar.timestamps (on page 461)

bufferVar.timestamps

This attribute contains the relative timestamp for each reading in the specified buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	bufferVar.clear()	Not applicable	Not applicable

Usage

timestamp = *bufferVar*.timestamps[*N*]

timestamp	The timestamp of reading number *N* in the specified reading buffer when the reading was acquired, relative to the first reading in the buffer
bufferVar	The reading buffer; can be a user-defined or default reading buffer
N	The reading number: 1 to *bufferVar*.n

Details

This buffer recall attribute is a Lua table that contains the timestamp of when each reading occurred. The timestamp is in seconds relative to the first measurement added to the reading buffer. It is collected at the beginning of the aperture time of each measurement.

The resolution of the timestamp is set by the *bufferVar*.timestampresolution attribute.

When used with the base timestamp attributes, you can calculate the absolute timestamp of each measurement.

Example

```
buffer = slot[1].smu[1].defbuffer2
print(buffer.timestamps[2])
printbuffer(1, buffer.n, buffer.timestamps)
```

Retrieve the timestamp of the second reading stored in defbuffer2 on channel 1 of the SMU in slot 1.

Then retrieve the timestamps of each reading stored in the same buffer.

Example output:

```
6.63871e-02
0.00000e+00, 6.63871e-02, 1.32775e-01, 1.99160e-01, 2.65548e-01, 3.31938e-01, 3.98327e-01, 4.64711e-01, 5.31093e-01, 5.97480e-01
```

This output indicates that the second measurement occurred 66.3871 ms after the first measurement, then lists the timestamp for each measurement in slot[1].smu[1].defbuffer2 relative to its first measurement.

Also see

bufferVar.basetimestampfractional (on page 444)
bufferVar.basetimestampseconds (on page 445)
bufferVar.clear() (on page 447)
bufferVar.n (on page 452)
bufferVar.readings (on page 453)
bufferVar.statuses (on page 459)
bufferVar.timestampresolution (on page 460)
Reading buffers (on page 96)

slot[Z].bank[X].meminfo()

This function returns memory information for a specified module bank.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

memory = slot[*Z*].bank[*X*].meminfo()

memory	Contains information about the memory usage for the specified memory bank
Z	Module slot number
X	Module memory bank number

Details

Use this command to retrieve a table of the allocated or used memory for a module bank.

The returned table contains the following fields:

- `total`: The total memory available for allocation
- `readingbuffer`: Memory in use by reading buffers
- `used`: The total memory in use
- `configlist`: Memory in use by configuration lists

Example 1

```
memory = slot[1].bank[1].meminfo()
for i, value in pairs(memory) do
print(i, value)end
```

Queries and retrieves all lists of memory values on bank 1 of the module installed in slot 1.

Output:

```
total:            1.53600e+05
readingbuffer:    2.17600e+03
used              2.17600e+03
configlist        0.00000e+00
```

Example 2

```
memory = slot[1].bank[1].meminfo()
print(memory.used)
```

Retrieves the total amount of memory used for bank 1 of the module installed in slot 1.

Output:

```
used              2.17600e+03
```

Also see

Memory considerations for the runtime environment (on page 42)

slot[Z].firmware.update()

This function updates the firmware of a module after the firmware file is copied to the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].firmware.update()
```

Z	Module slot number

Details

This command updates the module firmware. If the update is successful, the module is also restarted.

Before issuing this command, you need to load the firmware file onto the instrument. To download an image, use the mainframe `flash` and `endflash` keyword sequence or the `firmware.image.load()` command.

To update multiple modules with the same firmware file, you can send this command multiple times with different slot numbers.

For more information, see the applicable Reference Manual for your module, available from tek.com.

CAUTION: Do not turn off power until the update process is complete.

Example

```
firmware.image.load("/usb1/module/MSMU60-2-1.0.0.x")
slot[1].firmware.update()
```

Load the module firmware `MSMU60-2-1.0.0.x` file in the `module` folder of the USB drive inserted into the front of the mainframe.

Update the module in slot 1.

Also see

firmware.image.load() (on page 167)
slot[Z].firmware.verify (on page 463)

slot[Z].firmware.verify()

This function verifies that the firmware update succeeded.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
result = slot[Z].firmware.verify()
```

result	Status of the firmware verification: ▪ Firmware not correctly installed: `false` ▪ Firmware update succeeded: `true`
Z	Module slot number

Details

This function verifies that the firmware has updated correctly after you run the `slot[Z].firmware.update()` command.

This function compares the firmware downloaded to the mainframe and firmware in the module in the specified slot. If they are equal, the function returns `true`. Otherwise, it returns `false`.

You can use this command to determine that the firmware has flashed correctly after performing the `slot[Z].firmware.update()` command without downloading the image again.

Example

```
print(slot[1].firmware.verify())
```

Verifies that the firmware of the module installed in slot 1 updated correctly.

Also see

slot[Z].firmware.update (on page 463)

slot[Z].license

This attribute stores the license texts for the module installed in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

license = slot[*Z*].license

license	License text for the instrument
Z	Module slot number

Details

This attribute contains a large amount of text, including the Tektronix End User License Agreement and any open source software licenses in effect.

Example

```
print(slot[1].license)
```

Returns the text of the licenses for the module installed in slot 1 of the mainframe.

Also see

None

slot[Z].manufacturer

This attribute stores the manufacturer of the module in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Tektronix

Usage

manufacturer = slot[*Z*].manufacturer

manufacturer	This is always TEKTRONIX
Z	Module slot number

Example

```
print(slot[1].manufacturer)
```

Returns the manufacturer of the module installed in slot 1:

```
TEKTRONIX
```

Also see

None

slot[Z].model

This attribute stores the model number of the module in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not applicable	Not applicable

Usage

model = slot[*Z*].model

model	The model number of the module
Z	Module slot number

Example

```
print(slot[1].model)
```

Returns the model number of the module installed in slot 1.

Also see

None

slot[Z].revision

This attribute stores the firmware revision number of the module in the specified slot of the mainframe.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Firmware update	Nonvolatile memory	Not applicable

Usage

revisionNumber = slot[*Z*].revision

revisionNumber	The module firmware revision number
Z	Module slot number

Example

```
print(slot[1].revision)
```

Returns the firmware revision number of the module installed in slot 1.

Also see

None

slot[Z].serialno

This attribute stores the serial number of the module installed in the specified mainframe slot.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Nonvolatile memory	Not applicable

Usage

serialno = slot*[Z]*.serialno

serialno	The serial number of the module
Z	Module slot number

Example

```
print(slot[1].serialno)
```

Returns the serial number of the module installed in slot 1.

Also see

None

slot[Z].smu[X].abort()

This function terminates all overlapped operations on the specified channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].abort()`

Z	Module slot number
X	Module channel number

Details

This function terminates all overlapped operations on the specified channel.

This function does not turn the output off or change any settings.

Example

```
slot[1].smu[1].abort()
slot[1].smu[1].measure.count = 1000
slot[1].smu[1].measure.nplc = 1
slot[1].smu[1].measure.overlappedv(slot[1].smu[1].defbuffer1)
slot[1].smu[1].abort()
```

Aborts overlapped operations for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.overlappedY() (on page 500)

slot[Z].smu[X].acal.adjust()

This function runs the selected automatic calibration procedure.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].acal.adjust(procedure)`

Z	Module slot number
X	Module channel number
procedure	The automatic calibration procedure: ▪ Current source offset: `slot[Z].smu[X].ACAL_I_SOURCE_OFFSET` ▪ Current measure offset: `slot[Z].smu[X].ACAL_I_MEASURE_OFFSET` ▪ Use a known zero and reference voltage to characterize the auxiliary ADC: `slot[Z].smu[X].ACAL_AUX_ADC` ▪ Run all autocalibration procedures: `slot[Z].smu[X].ACAL_ADJUST_ALL`

Details

Runs the automatic calibration procedure specified by the function parameter.

`slot[Z].smu[X].ACAL_I_SOURCE_OFFSET`	Learns the ADC setting that outputs nearest to 0 A for each current source range, then adjusts the source calibration constants accordingly.
`slot[Z].smu[X].ACAL_I_MEASURE_OFFSET`	Applies 0 A to the current measure circuit and learns the offset of each range, then adjusts the measure calibration constants accordingly.
`slot[Z].smu[X].ACAL_AUX_ADC`	Uses a known zero and reference voltage to characterize the auxiliary ADC, then adjusts the ADC calibration constants accordingly. This ADC is used for the guard measurement and internal measurements.
`slot[Z].smu[X].ACAL_ADJUST_ALL`	Runs all autocalibration commands sequentially.

When the specified calibration procedure is completed, the adjusted calibration constants are independently saved for each procedure with the time and temperature. These constants are used to update the recommended due date for the procedure and to determine when the automatic calibration constants expire.

If a procedure fails, an error message is returned and the calibration data is not changed. If a failure occurs while running `slot[Z].smu[X].ACAL_ADJUST_ALL`, the data is not updated for any procedure.

Each procedure requires the user to put the SMU into a high impedance output state before running `slot[Z].smu[X].acal.adjust()`. This prevents the user from needing to disconnect the output leads when performing automatic calibration. An error message will be generated if the SMU is not in a high impedance state when the command runs.

When performing a standard calibration routine, the relevant automatic calibration adjustments are reset to nominal values, and the recommended automatic calibration due date is updated. For example, running `slot[Z].smu[X].measure.calibratei()` will reset the auto calibration data associated with `slot[Z].smu[X].ACAL_I_MEASURE_OFFSET`. For more information, see the *MP5000 Series SMU Calibration and Adjustment Manual*, available from tek.com.

Example

```
slot[3].smu[1].acal.adjust(slot[3].smu[1].ACAL_I_MEASURE_OFFSET)
```

Performs an automatic calibration adjustment procedure for current measure offset.

Also see

slot[Z].smu[X].acal.adjustdate() (on page 469)
slot[Z].smu[X].source.offmode (on page 523)

slot[Z].smu[X].acal.adjustdate()

This function returns the date when a calibration procedure was last run.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

date = slot[*Z*].smu[*X*].acal.adjustdate(*procedure*)

date	Date of last adjustment in seconds since January 1, 1970, 12:00 AM (UTC)
Z	Module slot number
X	Module channel number
procedure	The procedure to report a date for: ▪ slot[*Z*].smu[*X*].ACAL_I_SOURCE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_I_MEASURE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_AUX_ADC

Details

Returns the last run date of the specified automatic calibration procedure in seconds since January 1, 1970, 12:00 AM (UTC). This date is automatically updated after the specified procedure completes successfully.

Example

```
print(os.date('%m-%d-%Y %H:%M:%S', slot[3].smu[1].acal.adjustdate(slot[3].smu[1].ACAL_I_MEASURE_OFFSET)))
```

Returns the date and time of the last automatic calibration procedure for current measurement offset. For example:

```
05-20-2025 23:17:00
```

Also see

slot[Z].smu[X].acal.adjust()
slot[Z].smu[X].acal.due() (on page 470)

slot[Z].smu[X].acal.due()

This function returns the date that an automatic calibration procedure is due.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

date = slot[*Z*].smu[*X*].acal.due(*procedure*)

date	The calibration due date in seconds since January 1, 1970, 12:00 AM (UTC)
Z	Module slot number
X	Module channel number
procedure	Procedure to report due status for: ▪ slot[*Z*].smu[*X*].ACAL_I_SOURCE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_I_MEASURE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_AUX_ADC

Details

This function returns the due date of the specified automatic calibration procedure in seconds since January 1, 1970, 12:00 AM (UTC). The procedure should be run again after this time for best performance.

The slot[*Z*].smu[*X*].acal.valid() function returns false if the present time and date is later than this due date.

This date is automatically updated to 30 days after the specified procedure completes successfully.

Example

```
print(os.date('%m-%d-%Y %H:%M:%S', slot[3].smu[1].acal.due(slot[3].smu[1].ACAL_I_MEASURE_OFFSET)))
```

Returns the date and time that the next calibration procedure should be run for current measurement offset. For example:

```
06-03-2025 10:00:P00
```

Also see

slot[Z].smu[X].acal.adjust()
slot[Z].smu[X].acal.adjustdate() (on page 469)
slot[Z].smu[X].acal.valid() (on page 472)

slot[Z].smu[X].acal.temperaturedelta()

This function returns the temperature delta between the present and last time a procedure was run.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

temperature = slot[*Z*].smu[*X*].acal.temperaturedelta(*procedure*)

temperature	The temperature difference in degrees Celsius
Z	Module slot number
X	Module channel number
procedure	Procedure to report temperature delta for: ▪ slot[*Z*].smu[*X*].ACAL_I_SOURCE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_I_MEASURE_OFFSET ▪ slot[*Z*].smu[*X*].ACAL_AUX_ADC

Details

This function returns the difference between the present SMU temperature and the temperature recorded at the completion of the specified automatic calibration procedure.

The slot[*Z*].smu[*X*].acal.valid() function returns false if the difference is ± 5 °C from the last recorded temperature.

Example

```
print(tostring(slot[3].smu[1].acal.temperaturedelta(slot[3].smu[1].ACAL_I_MEASURE_
OFFSET)))
```

Might return the following result, indicating that the internal temperature of the SMU is presently approximately 5.9 degrees lower than it was the last time the current measurement offset automatic calibration procedure was run:

```
-5.9214274814984
```

Also see

slot[Z].smu[X].acal.adjust()
slot[Z].smu[X].acal.valid() (on page 472)

slot[Z].smu[X].acal.valid()

This function determines if the automatic calibration constants are valid.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

valid = slot[*Z*].smu[*X*].acal.valid(*procedure*)

valid	The validity of the constants: ▪ Valid: true ▪ Invalid: false
Z	Module slot number
X	Module channel number
procedure	The procedure to report constants for: ▪ Current source offset: slot[*Z*].smu[*X*].ACAL_I_SOURCE_OFFSET ▪ Current measure offset: slot[*Z*].smu[*X*].ACAL_I_MEASURE_OFFSET ▪ Auxiliary ADC: slot[*Z*].smu[*X*].ACAL_AUX_ADC

Details

This function determines if the calibration constants are valid. The function returns true if the constants are valid, and false if the constants have expired. If false, the specified procedure should be run.

Constants are valid within 30 days of the last autocalibration run, and within ± 5 °C of the last recorded autocalibration temperature.

To research a false return, you can use the slot[*Z*].smu[*X*].acal.adjustdate(), slot[*Z*].smu[*X*].acal.due(), and slot[*Z*].smu[*X*].acal.temperaturedelta() functions when slot[*Z*].smu[*X*].acal.valid() is run.

To check the status of autocalibration, you can use slot[*Z*].status.questionable.ACAL.

Example

```
print(slot[3].smu[1].acal.valid(slot[3].smu[1].ACAL_I_MEASURE_OFFSET))
```

In this example, the constants for the current measurement offset calibration procedure have expired, so the current measurement offset procedure should be run again.

Example output:

```
false
```

Also see

slot[Z].smu[X].acal.adjustdate() (on page 469)
slot[Z].smu[X].acal.due() (on page 470)
slot[Z].smu[X].acal.temperaturedelta() (on page 471)

slot[Z].smu[X].config.active()

This function returns a list of the active channel settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
config = slot[Z].smu[X].config.active()
```

config	A table of attributes representing the active configuration of the channels
Z	Module slot number
X	Module channel number

Details

Use this command to return a configuration of all active attributes on the selected channel. The configuration is represented by a Lua table. This configuration can be used to recall stored settings to a channel, added to an existing configuration list, or used as a reference when creating a new configuration list.

For additional details about the attributes stored in a configuration index, see Settings stored in a configuration index (on page 55).

```
slot[1].smu[1].configlist.create("MyConfigList")
ch1Config = slot[1].smu[1].config.active()
slot[1].smu[1].configlist.store("myConfigList", 3, ch1Config)
```

Creates a configuration list named MyConfigList.

Saves the attribute set on channel 1 of slot 1 to ch1Config.

Stores that configuration to the configuration list myConfigList in index 3.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].config.default() (on page 474)
slot[Z].smu[X].config.set() (on page 474)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.store() (on page 482)

slot[Z].smu[X].config.default()

This function recalls a configuration that represents the default channel settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

config = slot[*Z*].smu[*X*].config.default()

config	A set of attributes representing the default configuration
Z	Module slot number
X	Module channel number

Details

Use this command to return a table of the default configuration settings for a channel. The configuration is represented by a Lua table. This configuration can be used to recall stored settings to a channel, added to an existing configuration list or used as a reference when creating a new configuration list.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
startConfig = slot[1].smu[1].config.default()
```

Saves the default configuration for channel 1 of the module installed in slot 1 to the variable `startConfig`.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].config.active() (on page 473)
slot[Z].smu[X].config.set() (on page 474)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.store() (on page 482)

slot[Z].smu[X].config.set()

This function sets a channel configuration based on a table of attributes.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

slot[*Z*].smu[*X*].config.set(*config*)

Z	Module slot number
X	Module channel number
config	A table of attributes

Details

This function sets the active channel configuration based on a defined table. If the table is missing entries, those entries are not changed.

Example

```
slot[1].smu[1].config.set(setup1)
```

Configures channel 1 of the module installed in slot 1 to the settings in the configuration table `setup1`.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].config.active() (on page 473)
slot[Z].smu[X].config.default() (on page 474)

slot[Z].smu[X].configlist.append()

This function adds a step to the end of a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].configlist.append("configListName")
slot[Z].smu[X].configlist.append("configListName", config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
config	A table of attributes to be added to the configuration list

Details

Use this function to add an index to the end of a configuration list. If *config* is not specified, the active configuration is used.

Any attributes not specified in *config* will inherit the values specified by the reference configuration used when the configuration list was created.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].smu[1].configlist.append("MyConfigList",measureConfig2)
```

Adds the table of attributes in `measureConfig2` to the end of the configuration list `MyConfigLIst` on channel 1 of the module in slot 1.

Also see

slot[Z].smu[X].config.active() (on page 473)
slot[Z].smu[X].config.default() (on page 474)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 477)
slot[Z].smu[X].configlist.deletelist() (on page 478)

slot[Z].smu[X].configlist.create()

This function creates a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].configlist.create("configListName")
slot[Z].smu[X].configlist.create("configListName", config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list; must be unique on this channel
config	A table of attributes to be used as the reference configuration

Details

This command creates a new configuration list. If *config* is not specified, the active channel configuration will be used as the reference configuration. The reference configuration is used to fill in unspecified settings when additional configurations are added. To add configuration indexes to this list, use the `slot[Z].smu[X].configlist.store()` or `slot[Z].smu[X].configlist.append()` commands.

The *configListName* parameter must be unique on the specified slot and channel. If *configListName* specifies an existing configuration list, an error is generated. The original configuration list is not overwritten.

Configuration lists are not saved when the instrument is turned off. They are preserved through resets.

Example

```
slot[1].smu[1].configlist.create("MyConfigList", "drainConfig")
```

Creates a configuration list named `MyConfigList` based on the configuration `drainConfig` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].config.active() (on page 473)
slot[Z].smu[X].config.default() (on page 474)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.delete() (on page 477)
slot[Z].smu[X].configlist.store() (on page 482)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.delete()

This function deletes a step in a configuration list at the specified index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].configlist.delete("configListName", index)`

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list

Details

Deletes an index from a configuration list. An index must be specified.

When an index is deleted from a configuration list, the index numbers of the following indexes are shifted up by one. For example, if you have a configuration list with 10 indexes and you delete index 3, the index that was numbered 4 becomes index 3, and the following indexes are renumbered in sequence to index 9. Because of this, if you want to delete several nonconsecutive indexes in a configuration list, it is best to delete the highest numbered index first, then the next lowest index, and so on.

When deleting several consecutive indexes, it is best to delete the lowest number repeatedly until the indexes are removed.

To delete a configuration list, use the `slot[Z].smu[X].configlist.deletelist()` command.

The former functionality of this command is supported but deprecated beginning with mainframe firmware version 1.1.0. The former functionality allowed you to delete an entire configuration list by omitting the *index* parameter.

Example

```
slot[1].smu[1].configlist.delete("myConfigList", 2)
```

Deletes configuration index 2 from the configuration list named `myConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.deletelist() (on page 478)
slot[Z].smu[X].configlist.size() (on page 481)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.deletelist()

This function deletes a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].configlist.deletelist("configListName")`

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Details

Use this function to delete a configuration list.

Example

```
slot[1].smu[1].configlist.deletelist("myConfigList")
```

Deletes the configuration list named `myConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 477)

slot[Z].smu[X].configlist.query()

This function returns a full or partial configuration for the given step.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
config = slot[Z].smu[X].configlist.query("configListName", index)
config = slot[Z].smu[X].configlist.query("configListName", index, diff)
```

config	A table that contains all channel attributes or only attributes that are different from the reference configuration at the given index
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number that defines a configuration index in the configuration list
diff	A boolean value. `True` returns only the differences in the selected configuration from the reference configuration. `False` returns the full selected configuration.

Details

This function returns a configuration table for the given step. If *diff* is not provided or is set to `false`, the table that is returned contains all attributes. If *diff* is true, only the attributes that are different than the reference configuration for the named configuration list are returned.

Example

```
config = slot[Z].smu[X].configlist.query("myConfigList", 5)
```

Returns all configuration settings saved in index 5 of the configuration list named `myConfigList`.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.reference() (on page 481)
slot[Z].smu[X].configlist.store() (on page 482)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.recall()

This function recalls and applies the configuration at a specified index to the channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].configlist.recall("configListName", index)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list

Details

Use this command to recall the settings stored in a specific configuration index in a configuration list. These recalled settings are immediately applied to the channel. The active configuration of the instrument channel is modified to match the recalled configuration. A configuration can be called if the channel output is on or off. Each index contains most of the measure and source attributes of the channel.

An error message is generated:

- If you do not specify an index.
- If you specify an invalid index (for example, calling index 3 when there are only two indexes in the configuration list).
- If you try to recall an index from an empty configuration list.

For additional details about the information this command recalls when using a configuration list query command, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].smu[1].configlist.recall("MyConfigList", 5)
```

Recalls and applies configuration index 5 in a configuration list named `MyConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 478)
slot[Z].smu[X}.configlist.query() (on page 479)
slot[Z].smu[X].configlist.size() (on page 481)
slot[Z].smu[X].configlist.store() (on page 482)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.reference()

This function returns a table of the reference configuration used when the specified configuration list on the channel was created.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

config = slot[*Z*].smu[*X*].configlist.reference("*configListName*")

config	A table containing the attributes that represent the reference configuration
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Details

Use this function to return the reference configuration used to create the specified configuration list on a channel. This configuration is returned as a Lua table.

Example

```
referenceConfig = slot[1].smu[1].configlist.reference("myConfigList")
```

Returns the reference configuration of `myConfigList` on channel 1 of the module in slot 1 to `referenceConfig`.

Also see

slot[Z].smu[X].configlist.create() (on page 476)

slot[Z].smu[X].configlist.size()

This function returns the number of configuration indexes in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

indexCount = slot[*Z*].smu[*X*].configlist.size("*configListName*")

indexCount	A number that represents the total count of indexes stored in the specified configuration list
Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list

Example

```
print(slot[1].smu[1].configlist.size("testMeasList"))
```

Returns the number of configuration indexes in a configuration list named `testMeasList` on channel 1 of the module installed in slot 1.

Example output:

```
1
```

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 478)
slot[Z].smu[X].configlist.query() (on page 479)
slot[Z].smu[X].configlist.recall() (on page 480)
slot[Z].smu[X].configlist.store() (on page 482)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.store()

This function stores a configuration into the named configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].configlist.store("configListName")
slot[Z].smu[X].configlist.store("configListName", index)
slot[Z].smu[X].configlist.store("configListName", index, config)
```

Z	Module slot number
X	Module channel number
configListName	A string that represents the name of a configuration list
index	A number starting from 1 that defines a specific configuration index in the configuration list
config	A table of channel attributes

Details

Use this command to store a set of attributes to a configuration list at the specified index. If no configuration is provided, the active settings on the specified channel and slot are used. If a partial configuration is used, the missing values are taken from the reference configuration for the configuration list.

If the index already contains a configuration, that configuration is overwritten. If the index is the number of steps plus 1 or not specified, the configuration is saved at the end of the list. An error is generated if the index is greater than 1 more than the number of steps already present in the configuration list. To always add the configuration to the end of the configuration list, use `slot[Z].smu[X].configlist.append()`.

Configuration lists are not saved when the instrument is turned off. They are preserved through a reset.

The former functionality of this command is supported but deprecated, beginning with mainframe firmware version 1.1.0. Use this command to store a configuration into a named configuration list and `slot[Z].smu[X].configlist.append()` to add steps to an existing configuration list.

For additional details about the attributes stored in a configuration list, see Settings stored in a configuration index (on page 55).

Example

```
slot[1].smu[1].configlist.store("MyConfigList")
```

Stores the active settings of the instrument to the end of the configuration list `MyConfigList` on channel 1 of the module installed in slot 1.

```
slot[1].smu[1].configlist.store("MyConfigList", 5)
```

Stores the active settings of the instrument to configuration index 5 in the configuration list `MyConfigList` on channel 1 of the module installed in slot 1.

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.append() (on page 475)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 478)
slot[Z].smu[X].configlist.query() (on page 479)
slot[Z].smu[X].configlist.recall() (on page 480)
slot[Z].smu[X].configlist.reference() (on page 481)
slot[Z].smu[X].configlist.size() (on page 481)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].smu[X].configlist.table()

This function returns the names of configuration lists stored on the channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
configListTable = slot[Z].smu[X].configlist.table()
```

configListTable	A table that contains the names of the configuration lists stored on the specified slot and channel
Z	Module slot number
X	Module channel number

Details

This function returns a Lua table that contains the names of configuration lists that are stored on the specified channel.

To see the names of the configuration lists that have been created, use either of the code examples under Example 1 and Example 2. If no configuration lists exist, the table is empty.

Example 1

```
-- Create configuration lists.
slot[1].smu[1].configlist.create("configlist1")
slot[1].smu[1].configlist.create("configlist2")
-- Retrieve the names of the lists.
myConfigLists = slot[1].smu[1].configlist.table()
for i, name in pairs(myConfigLists) do
    print(name)
end
```

Create configuration lists.

Retrieve the names of all configuration lists:

```
configlist1
configlist2
```

Example 2

```
-- Create configuration lists.
slot[1].smu[1].configlist.create("configlist1")
slot[1].smu[1].configlist.create("configlist2")
-- Retrieve the names of the lists.
printbuffer(1, table.getn(myConfigLists), myConfigLists)
end
```

Create configuration lists.

Retrieve the names of all configuration lists:

```
configlist1
configlist2
```

Also see

Configuration lists (on page 50)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].smu[X].configlist.delete() (on page 478)
slot[Z].smu[X].configlist.recall() (on page 480)
slot[Z].smu[X].configlist.query() (on page 479)
slot[Z].smu[X].configlist.size() (on page 481)
slot[Z].smu[X].configlist.store() (on page 482)

slot[Z].smu[X].contact.check()

This function performs a contact check measurement and compares the result to the user-specified contact check threshold setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

contactCheckResult, *hi*, *lo* = slot[*Z*].smu[*X*].contact.check()

contactCheckResult	The result of the contact check measurement: `true` or `false`
hi	The contact check hi measurement in ohms; only available with calibration unlocked
lo	The contact check lo measurement in ohms; only available with calibration unlocked
Z	Module slot number
X	Module channel number

Details

The contact check function identifies excessive resistance in the force or sense leads when making 4-wire remote measurements.

Performs a contact check measurement sequence, then compares the result to the user-specified threshold set using `slot[Z].smu[X].contact.threshold`. Returns `true` if the measurement is within the threshold. If it is not within the threshold, it returns `false`.

The command also returns the resistance measurements used in determining the contact check result, but only when calibration is unlocked. These measurements are needed for the contact check calibration procedure.

You cannot use the high impedance output or output mode setting with contact check.

Example

```
print(slot[1].smu[1].contact.check())
```

Returns `true` or `false` based on the contact resistances for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].contact.r() (on page 485)
slot[Z].smu[X].contact.threshold (on page 487)
slot[Z].smu[X].contact.speed (on page 486)
slot[Z].smu[X].source.offmode (on page 523)
slot[Z].smu[X].source.output (on page 524)

slot[Z].smu[X].contact.r()

This function performs a contact check measurement and returns the measured contact resistances.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

hi, lo = `slot[Z].smu[X].contact.r()`

hi	The contact check high measurement (between HI and SHI) in ohms
lo	The contact check low measurement (between LO and SLO) in ohms
Z	Module slot number
X	Module channel number

Example

```
print(slot[1].smu[1].contact.r())
```

Performs a measurement of the high and low contact resistances for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].contact.check() (on page 484)
slot[Z].smu[X].contact.speed (on page 486)

slot[Z].smu[X].contact.speed

This attribute sets the contact check measurement speed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	CONTACT_NORMAL

Usage

```
slot[Z].smu[X].contact.speed = contactCheckSpeed
contactCheckSpeed = slot[Z].smu[X].contact.speed
```

Z	Module slot number
X	Module channel number
contactCheckSpeed	The contact check speed option: ▪ `slot[Z].smu[X].CONTACT_NORMAL` ▪ `slot[Z].smu[X].CONTACT_SLOW` ▪ `slot[Z].smu[X].CONTACT_CAL_SPEED`

Details

This attribute contains the contact check speed normal, slow, or calibration speed. Normal measurement speed is approximately 5 ms. Slow is approximately 53 ms. The speed setting affects the contact check function and the contact resistance measurement function.

The speed setting directly changes the number of conversions averaged during each phase of a contact check measurement.

Example

```
slot[1].smu[1].contact.speed = slot[1].smu[1].CONTACT_SLOW
```

Sets the contact check measurement speed to slow for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].contact.r() (on page 485)
slot[Z].smu[X].contact.check() (on page 484)

slot[Z].smu[X].contact.threshold

This attribute sets the contact check pass or fail threshold.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	50.0 (Ω)

Usage

```
slot[Z].smu[X].contact.threshold = contactCheckThreshold
contactCheckThreshold = slot[Z].smu[X].contact.threshold
```

Z	Module slot number
X	Module channel number
contactCheckThreshold	The contact check pass or fail threshold in ohms

Details

This attribute contains the contact check pass/fail threshold used by the contact check function `slot[Z].smu[X].contact.check()`. Acceptable values are from 0 Ω to 300 Ω.

The threshold also effectively sets the contact check measure range used by `slot[Z].smu[X].contact.check()` and calibration.

Example

```
slot[1].smu[1].contact.threshold = 100
```

Sets the contact check function pass or fail threshold to 100 Ω for the module installed in slot 1.

Also see

slot[Z].smu[X].contact.check() (on page 484)

slot[Z].smu[X].contact.zero.adjust()

This function runs the contact check zero procedure.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].contact.zero.adjust(procedure)
```

Z	Module slot number
X	Module channel number
procedure	`slot[Z].smu[X].CONTACT_HI`

Details

This function runs the contact check hi zero procedure. This function automatically corrects for any offset in the contact check hi measurement circuit and applies the correction to the contact check function.

Upon successful completion, the adjustment is saved along with the present time and temperature. These are used to update the recommended calibration due date and determine whether the adjustment is valid.

If the correction fails, an error message is returned, and the adjustment data is not changed.

This function requires the user to remove all output connections before beginning, otherwise an error will be returned or an invalid adjustment will result.

Running `slot[Z].smu[X].contact.calibratehi()` will reset the adjustment data.

Example

```
slot[3].smu[1].contact.zero.adjust(slot[3].smu[1].CONTACT_HI)
```

Corrects the contact check hi measurement for channel 1 of the module installed in slot 3.

Also see

slot[Z].smu[X].contact.zero.adjustdate() (on page 488)

slot[Z].smu[X].contact.zero.adjustdate()

This function returns the last run date and time of the contact check zero procedure.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
date = slot[Z].smu[X].contact.zero.adjustdate(procedure)
```

date	The last run date of the contact check zero procedure in seconds since January 1, 1970, 12:00 AM (UTC)
Z	Module slot number
X	Module channel number
procedure	`slot[Z].smu[X].CONTACT_HI`

Details

This function returns the last run date and time of the contact check zero procedure in seconds since January 1, 1970, 12:00 AM (UTC). This date is automatically updated after successful completion of `slot[Z].smu[X].contact.zero.adjust()`.

Example

```
print(os.date('%m-%d-%Y %H:%M:%S', slot[3].smu[1].contact.zero.adjustdate(slot[3].
smu[1].CONTACT_HI)))
```

Returns the last run date and time of the contact check zero adjust procedure for channel 1 of the module installed in slot 3.

Also see

slot[Z].smu[X].contact.zero.adjust() (on page 487)
slot[Z].smu[X].contact.zero.due() (on page 489)

slot[Z].smu[X].contact.zero.due()

This function returns the date and time that the contact check zero procedure is due.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

date = slot[*Z*].smu[*X*].contact.zero.due(*procedure*)

date	Date and time that the contact check zero procedure is due
Z	Module slot number
X	Module channel number
procedure	slot[*Z*].smu[*X*].CONTACT_HI

Details

Returns the date and time of the next due contact check zero adjust procedure.

The slot[*Z*].smu[*X*].contact.zero.valid() function returns false if the current time and date is later than this due date.

This date is automatically updated after successful completion of slot[*Z*].smu[*X*].contact.zero.adjust().

Example

```
print(os.date('%m-%d-%Y %H:%M:%S', slot[3].smu[1].acal.adjustdate(slot[3].smu[1].CONTACT_HI)))
```

Might return

```
05-22-2026 00:38:00
```

Indicating that the due date and time of the next contact check zero adjust procedure for channel 1 of the module installed in slot 3 is May 22, 2026, at 12:38 AM.

Also see

slot[Z].smu[X].contact.zero.adjust() (on page 487)
slot[Z].smu[X].contact.zero.adjustdate() (on page 488)
slot[Z].smu[X].contact.zero.valid() (on page 491)

slot[Z].smu[X].contact.zero.temperaturedelta()

This function returns the difference between the present SMU temperature and the temperature of the SMU the last time the contact check zero adjust procedure was run.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

date = slot[*Z*].smu[*X*].contact.zero.temperaturedelta(*procedure*)

date	Temperature difference in degrees Celsius
Z	Module slot number
X	Module channel number
procedure	slot[*Z*].smu[*X*].CONTACT_HI

Details

This function returns the difference between the current SMU temperature and the temperature recorded at the last completion of slot[*Z*].smu[*X*].contact.zero.adjust().

The slot[*Z*].smu[*X*].contact.zero.valid() function returns false if this difference is outside of the allowed window.

Example

```
print(tostring(slot[3].smu[1].acal.temperaturedelta(slot[3].smu[1].CONTACT_HI)))
```

Might return

```
2.4
```

Indicating that the internal temperature of channel 1 of the module installed in slot 3 is 2.4 degrees Celsius higher than when the slot[Z].smu[X].contact.zero.adjust() command was last run.

Also see

slot[Z].smu[X].contact.zero.adjust() (on page 487)
slot[Z].smu[X].contact.zero.valid() (on page 491)

slot[Z].smu[X].contact.zero.valid()

This function returns the state of the contact check zero adjustment procedure.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

validity = slot[*Z*].smu[*X*].contact.zero.valid(*procedure*)

validity	The state of the contact check zero adjustment procedure. If calibration constants are valid, true. Otherwise, false.
Z	Module slot number
X	Module channel number
procedure	slot[*Z*].smu[*X*].CONTACT_HI

Details

This function returns true if the contact check zero adjustments are valid and false if expired. If false, run the slot[*Z*].smu[*X*].CONTACT_HI procedure again.

The following commands can also be used to investigate a false return for this function:

- slot[*Z*].smu[*X*].contact.zero.adjustdate()
- slot[*Z*].smu[*X*].contact.zero.due()
- slot[*Z*].smu[*X*].contact.zero.temperaturedelta()

Example

```
print(slot[3].smu[1].contact.zero.valid(slot[3].smu[1].CONTACT_HI))
```

Might return

```
true
```

Indicating that the contact check zero adjustments for channel 1 of module installed in slot 3 are valid.

Also see

slot[Z].smu[X].contact.zero.adjustdate() (on page 488)
slot[Z].smu[X].contact.zero.due() (on page 489)
slot[Z].smu[X].contact.zero.temperaturedelta() (on page 490)

slot[Z].smu[X].defbufferY

This attribute contains the default reading buffers.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle slot[Z].smu[X].defbuffer1.clear()	Not applicable	Not applicable

Usage

```
bufferContents = slot[Z].smu[X].defbufferY
bufferContents = slot[Z].smu[X].defbufferY[N]
```

bufferContents	The readings in the default reading buffer
Z	Module slot number
X	Module channel
Y	Default buffer number: `1` or `2`
N	Index of the reading in the buffer

Details

Each SMU channel contains two default reading buffers: `slot[Z].smu[X].defbuffer1` and `slot[Z].smu[X].defbuffer2`.

All routines that return measurements can also store them in either reading buffer. Overlapped measurements are always stored in a reading buffer. Synchronous measurements return either a single-point measurement or are stored in a reading buffer if they are passed to the measurement command.

Example

```
slot[1].smu[2].measure.count = 10
slot[1].smu[2].measure.v(slot[1].smu[2].defbuffer1)
format.data = format.ASCII
format.asciiprecision = 3
printbuffer(1, slot[1].smu[2].defbuffer1.n, slot[1].smu[2].defbuffer1.readings)
```

Set the measurement count to 10. Make and store voltage measurements in `defbuffer1` of channel 2 of the module installed in slot 1. Set the data format to ASCII with precision of 6, then return the information from `defbuffer1`. The use of `defbuffer1.n` (*bufferVar*`.n`) indicates that the instrument should output all readings in the reading buffer. Example output:

```
2.07e-05, 2.13e-05, 2.12e-05, 2.09e-05, 2.07e-05, 2.12e-05, 2.09e-05, 2.11e-05, 2.07e-05, 2.09e-05
```

Also see

None

slot[Z].smu[X].guard.v()

This function measures the guard output voltage.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

reading = slot[*Z*].smu[*X*].guard.v()

reading	The measurement value
Z	Module slot number
X	Module channel number

Details

The result of this function can help you determine if the guard is shorted or otherwise loaded in a way that makes it ineffective. This function measures the voltage of guard relative to SMU ground. It should be near zero when the current source range (or current limit range) is 10 µA or less. At higher output currents, the return value is not zero. The guard measurement is intended to be used as a diagnostic tool and does not produce a precise measurement. For example, if guard is shorted externally and is therefore ineffective, a large voltage error is observed when the SMU is on a low current range.

Example

```
GuardVoltage = slot[1].smu[1].guard.v()
print(GuardVoltage)
```

Measures the guard output voltage of channel 1 of the module installed in slot 1.

Also see

None

slot[Z].smu[X].makebuffer()

This function creates a user-defined reading buffer.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

bufferVar = slot[*Z*].smu[*X*].makebuffer(*bufferSize*)

bufferVar	The variable name of the new reading buffer
Z	Module slot number
X	Module channel number
bufferSize	The size of the reading buffer

Details

Reading buffers can be created and allocated dynamically using this function. Use *bufferSize* to designate the number of readings the buffer can store.

An error is returned if *bufferSize* exceeds the maximum storage capacity of the module. The available storage capacity varies and depends on how much memory is consumed by other reading buffers, configuration lists, trigger models, and sweep configurations.

Dynamically allocated reading buffers can be used interchangeably with the dedicated `slot[Z].smu[X].defbufferY` buffers.

To delete a reading buffer, set all references to the reading buffer equal to `nil`, then run the garbage collector.

Example

```
mybuffer2 = slot[1].smu[2].makebuffer(200)
```

Creates a 200-element reading buffer (`mybuffer2`) for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].defbufferY (on page 492)

slot[Z].smu[X].measure.aperture

This attribute configures the measurement aperture for a channel.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].measure.nplc	Configuration list index	1.66667e-2 (60 Hz line) 2.0e-02 (50 Hz line)

Usage

```
value = slot[Z].smu[X].measure.aperture
slot[Z].smu[X].measure.aperture = value
```

value	Aperture: `1e-6` to `500e-3` s
Z	Module slot number
X	Module channel number

Details

This sets a fixed measurement aperture time. Setting this attribute changes the `slot[Z].smu[X].measure.nplc` attribute.

To set an aperture that is a multiple of power line cycles, use `slot[Z].smu[X].measure.nplc`.

Example

```
slot[1].smu[2].measure.aperture = 2e-3
print(slot[1].smu[2].measure.aperture)
```

Set the measurement aperture to 2 ms for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.nplc (on page 499)

slot[Z].smu[X].measure.autorangeY

This attribute contains the state of the measure autorange control (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].measure.rangeY	Configuration list index	slot[Z].smu[X].ON

Usage

```
measureAutorange = slot[Z].smu[X].measure.autorangeY
slot[Z].smu[X].source.autorangeY = measureAutorange
```

measureAutorange	State of autorange: ▪ Disabled (fixed range): `slot[Z].smu[X].OFF` ▪ Enabled (autorange): `slot[Z].smu[X].ON`
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This attribute indicates the measurement autorange state.

Setting this attribute to `slot[Z].smu[X].OFF` puts the SMU on a fixed measure range. The fixed range is the present SMU measure range.

Setting this attribute to `slot[Z].smu[X].ON` puts the SMU measure circuit in autorange mode. It remains on its present measure range until the next measurement is requested. At this time, a new range may be selected based on the result of the measurement. The `slot[Z].smu[X].measure.rangeY` attribute is updated to reflect the new range.

The low range limit for measurements sets the lowest range that the instrument uses when autorange is on. This feature minimizes autorange settling times when measurements require numerous range changes.

When autorange is on, multiple measurements may be made to determine the best measurement range. This can result in longer measurement times. For the most consistent timing interval, use fixed measurement ranges.

Example

```
slot[2].smu[1].measure.autorangei = slot[2].smu[1].ON
```

Enables autorange for the current measurement function for channel 1 of the module installed in slot 2.

Also see

slot[Z].smu[X].measure.lowrangeY (on page 498)
slot[Z].smu[X].measure.rangeY (on page 501)

slot[Z].smu[X].measure.count

This attribute sets the number of measurements made when a measurement is requested.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	1

Usage

```
count = slot[Z].smu[X].measure.count
slot[Z].smu[X].measure.count = count
```

count	Number of measurements
Z	Module slot number
X	Module channel number

Details

This attribute controls the number of measurements made and stored when a measurement is requested to be stored in a reading buffer. If a reading buffer is not specified, this attribute is ignored and one measurement is made.

If the count exceeds the remaining capacity of the reading buffer, no measurements are made.

If the count is set to a value greater than 1, any measurement delay set by `slot[Z].smu[X].measure.delay` only occurs before the first measurement. To control the interval between successive measurements, use `slot[Z].smu[X].measure.interval`.

Example

```
slot[1].smu[1].measure.count = 100
```

Sets the number of measurements to be performed on channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.overlappedY() (on page 500)
slot[Z].smu[X].measure.Y() (on page 504)

slot[Z].smu[X].measure.delay

This attribute controls the measurement delay.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0 s

Usage

```
delay = slot[Z].smu[X].measure.delay
slot[Z].smu[X].measure.delay = delay
```

delay	The measure delay value (in seconds): 0 to 57
Z	Module slot number
X	Module channel number

Details

This attribute allows for additional delay (settling time) before making a measurement.

If `slot[Z].smu[X].measure.count` is greater than 1, the measurement delay is only added to the first measurement.

To disable all measurement delays, set `slot[Z].smu[X].measure.delay` to `0`.

Example

```
slot[1].smu[1].measure.delay = 0.2
```

Sets the first measurement delay to 200 ms for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.interval (on page 497)

slot[Z].smu[X].measure.interval

This attribute sets the interval between multiple measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0

Usage

```
interval = slot[Z].smu[X].measure.interval
slot[Z].smu[X].measure.interval = interval
```

interval	The interval value (in seconds): 0 to 57
Z	Module slot number
X	Module channel number

Details

This attribute sets the time interval between measurements when `slot[Z].smu[X].measure.count` is set to a value greater than 1. The SMU attempts to start each measurement when scheduled. If the SMU cannot keep up with the interval setting, measurements are made as quickly as possible.

Example

```
slot[1].smu[1].measure.interval = 100e-3
```

Sets the time between measurements to 100 ms for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.count (on page 496)

slot[Z].smu[X].measure.lowrangeY

This attribute sets the lowest measurement range to be used when the instrument is autoranging.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	Voltage: 200e-3 (200 mV) Current: 1e-6 (1 µA)

Usage

measureLowrange = slot[*Z*].smu[*X*].measure.lowrange*Y*
slot[*Z*].smu[*X*].source.lowrange*Y* = *measureLowrange*

measureLowrange	Set to the maximum expected voltage or current: **MSMU60-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 60 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 1.5 A **MSMU200-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 200 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 7 A
Z	Module slot number
X	Module channel number
Y	The source function; v = voltage, i = current

Details

This attribute is used with autoranging to put a lower boundary on the range used. Since lower ranges generally require greater settling times, setting a lowest range limit could make measurements require less settling time.

If the instrument is set to autorange and it is on a range lower than the specified low range, the range is changed to the *measureLowrange* range value when the next measurement is made.

The 7 A range on the MSMU200-2 is pulsing-only above 1.5 A. For more information, see the *MP5000 Series Source-Measure Unit Reference Manual*.

Example

```
slot[3].smu[1].measure.lowrangev = 6
```

Sets the low range for the voltage measurement function to 6 V for channel 1 of the module installed in slot 3.

Also see

slot[Z].smu[X].measure.autorangeY (on page 495)
slot[Z].smu[X].measure.rangeY (on page 501)

slot[Z].smu[X].measure.nplc

This command sets the time that the input signal is measured in number of power line cycles.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].measure.aperture	Configuration list index	1.0

Usage

nplc = slot[*Z*].smu[*X*].measure.nplc

nplc	Power line cycles: 0.00006 to 30
Z	Module slot number
X	Module channel number

Details

This setting configures the measurement aperture in number of power line cycles (NPLC) for the selected channel. The measurement aperture is the amount of time that the input signal is measured. The shortest amount of time results in the fastest reading rate but increases the reading noise. The longest amount of time provides the lowest reading noise but has the slowest reading rate. Settings between the fastest and slowest number of power line cycles are a compromise between speed and noise.

Each PLC for a line frequency of 60 Hz is 16.67 ms (1/60) and each PLC for a line frequency of 50 Hz is 20 ms (1/50). For 60 Hz, if you set the NPLC to 0.1, the measure time is 1.667 ms. However, the SMU uses a sampling analog-to-digital converter (ADC) that has a maximum sample rate of 1 MSa/s. Therefore, the aperture can only be set to the nearest microsecond. If you are connected to a power line with a frequency of 60 Hz, setting the NPLC to 1 PLC actually sets an aperture of 17 ms.

If you query the aperture setting using print(slot[*Z*].smu[*X*].measure.aperture), it always returns the SMU calculated value as the nearest aperture setting.

If you change this attribute, the slot[*Z*].smu[*X*].measure.aperture attribute is changed to the equivalent value of seconds, rounded to the nearest microsecond.

To set a fixed aperture in seconds, use slot[*Z*].smu[*X*].measure.aperture.

Example

```
slot[1].smu[2].measure.nplc = 0.12
NPLC1 = slot[1].smu[2].measure.nplc
print(NPLC1)
```

Reads the measurement aperture in NPLCs for channel 2 of the module installed in slot 1 and prints the value of NPLC1.

Also see

slot[Z].smu[X].measure.aperture (on page 494)

slot[Z].smu[X].measure.overlappedY()

This function starts an asynchronous (background) measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

```
slot[Z].smu[X].measure.overlappedY(rbuffer)
slot[Z].smu[X].measure.overlappediv(ibuffer, vbuffer)
```

Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
rbuffer	A reading buffer object where readings are stored
ibuffer	A reading buffer object where current readings are stored
vbuffer	A reading buffer object where voltage readings are stored

Details

This function starts a measurement and immediately returns the readings. The measurements are stored in a reading buffer, along with any ancillary information also acquired. If the instrument is configured to return multiple readings, the readings become available as they are made.

Measurements are in the following units: `v` = volts, `i` = amperes, `r` = ohms, `p` = watts.

The `slot[Z].smu[X].measure.overlappediv()` function stores current readings in *ibuffer* and voltage readings in *vbuffer*.

This function is an overlapped command. Script execution continues while measurements are made in the background. Attempts to access result values that have not yet been generated cause the script to wait for the data to become available. The `waitcomplete()` function can also be used to wait for the measurements to complete before continuing with the script.

Any data in a reading buffer is deleted before recording any measurements unless the reading buffer has been configured to append data.

Example

```
slot[1].smu[1].measure.overlappedv(slot[1].smu[1].defbuffer1)
```

Starts background voltage measurements for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.aperture (on page 494)

slot[Z].smu[X].measure.rangeY

This attribute contains the positive full-scale value of the measurement range for voltage or current.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].measure.autorangeY	Configuration list index	Voltage: 200 mV Current: 1 µA

Usage

```
rangeValue = slot[Z].smu[X].measure.rangeY
slot[Z].smu[X].measure.rangeY = rangeValue
```

rangeValue	Set to the maximum expected voltage or current: **MSMU60-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 60 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 1.5 A **MSMU200-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 200 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 7 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

The measure range determines the full-scale measurement input for the measurement. It also affects the accuracy of the measurements and the maximum signal that can be measured.

The highest available measurement range is determined by the source limit value. The 7 A range on the MSMU200-2 is pulsing-only above 1.5 A. For more information, see the *MP5000 Series Source-Measure Unit Reference Manual*.

You can set the measurement range to a fixed value or to automatic measure range (autorange). The range defaults to autorange for all functions. If you set a fixed range, autorange is set to off. If you set autorange on, the fixed range is in place until a measurement is made, at which time the range is set to accommodate the new measurement.

When a fixed range is selected, the range value is static. To ensure the best accuracy and resolution, use the lowest range possible that does not cause an overflow event.

Changing the range settings when the output is off does not update the hardware settings.

The combination of voltage and current ranges cannot exceed the maximum power of the instrument (30 W). For example, with a 20 V source range, the highest current measure range is 1.5 mA.

In specific source and measure cases, current measurement ranges are locked to the current source range or off-limit current source range. A new current range can be specified and will be retained, but a query will return the locked values. Voltage measurement ranges are not locked and a query will return the specified range.

Use the following table to determine if a specified source and measure setup falls into one of these cases:

Source	Output	Current measurement range	Voltage measurement range
Current	On	Locked to source current range value	Unlocked
Current	Off	Locked to source current range value	Unlocked
Voltage	Off	Locked to off-limit current range value	Unlocked
Voltage	On	Unlocked	Unlocked

Example

```
slot[1].smu[2].measure.rangev = 2
```

Selects the 2 V measurement range for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.autorangeY (on page 495)
slot[Z].smu[X].source.rangeY (on page 528)

slot[Z].smu[X].measure.rel.enableY

This attribute turns relative measurements on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].DISABLE

Usage

```
relEnable = slot[Z].smu[X].measure.rel.enableY
slot[Z].smu[X].measure.rel.enableY = relEnable
```

relEnable	Relative measurement control; set *relEnable* to one of the following values: ▪ Disable relative measurements: `slot[Z].smu[X].DISABLE` ▪ Enable relative measurements: `slot[Z].smu[X].ENABLE`
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power

Details

This attribute enables or disables relative measurements. When relative measurements are enabled, all subsequent measured readings are offset by the relative offset value specified by `slot[Z].smu[X].measure.rel.levelY`.

Each returned measured relative reading is the result of the following:

Relative reading = Actual measured reading − Relative offset value

Example

```
slot[1].smu[2].measure.rel.enablev = slot[1].smu[2].ENABLE
```

Enables relative voltage measurements for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.rel.levelY (on page 503)

slot[Z].smu[X].measure.rel.levelY

This attribute specifies the offset value for relative measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0

Usage

relValue = slot[*Z*].
smu[*X*].measure.rel.level*Y*
slot[*Z*].smu[*X*].measure.rel.level*Y* = *relValue*

relValue	Relative measurement offset value
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power

Details

When relative measurements are enabled (`slot[`*Z*`].smu[`*X*`].measure.rel.enable`*Y*), all subsequent measured readings are offset by the value of this attribute.

Each returned measured relative reading is the result of the following calculation:

Relative reading = Actual measured reading − Relative offset value

Example

```
slot[1].smu[2].measure.rel.levelv = slot[1].smu[2].measure.v()
```

Makes a voltage measurement using channel 2 of the module installed in slot 1 and then uses the reading as the relative offset value.

Also see

slot[Z].smu[X].measure.rel.enableY (on page 502)

slot[Z].smu[X].measure.Y()

This function makes one or more measurements.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
reading = slot[Z].smu[X].measure.Y()
reading = slot[Z].smu[X].measure.Y(readingBuffer)
iReading, vReading = slot[Z].smu[X].measure.iv()
iReading, vReading = slot[Z].smu[X].measure.iv(iReadingBuffer, vReadingBuffer)
```

reading	The last or only measured value returned; see **Details**
Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
readingBuffer	The reading buffer; can be a user-defined or default reading buffer
compValue	The compensation value used to calibrate current measure
iReading	The last or only current measurement value returned
vReading	The last or only voltage measurement value returned
iReadingBuffer	Reading buffer where current readings are stored; can be a user-defined buffer or a default reading buffer
vReadingBuffer	Reading buffer where voltage readings are stored; can be a user-defined buffer or a default reading buffer

Details

This function initiates one or more synchronous measurements based on the parameters specified above.

Measurements are in the following units of measure:

- `v` = volts
- `i` = amperes
- `r` = ohms
- `p` = watts

If you use this function without specifying a reading buffer, it makes one measurement and returns that measurement as *reading*. To use the additional information that is acquired while making a measurement or to return multiple readings, specify a reading buffer. If the instrument is configured to return multiple readings for a measurement and *readingBuffer* is specified, all readings are available in *readingBuffer*, but only the last measurement is returned as *reading*.

The `slot[Z].smu[X].measure.iv()` function returns the last actual current measurement and voltage measurement as iReading and vReading, respectively. Additionally, it can store current and voltage readings if buffers are provided *(iReadingBuffer* and *vReadingBuffer*).

The `slot[Z].smu[X].measure.count` attribute determines how many measurements are performed. When a reading buffer is used, it also determines the number of readings to store in the buffer. If a reading buffer is not specified, the SMU ignores the `slot[Z].smu[X].measure.count` attribute and only makes one measurement.

The *readingBuffer* is cleared before making any measurements unless the buffer is configured to append data.

Example

```
voltageValue = slot[1].smu[2].measure.v()
print(voltageValue)
```

Measures the voltage on channel 2 of the module installed in slot 1 and stores it in the `voltageValue` variable.

Also see

slot[Z].smu[X].measure.count (on page 496)
slot[Z].smu[X].measure.rel.enableY (on page 502)

slot[Z].smu[X].overlapped

This attribute contains the state of the overlapped mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].DISABLE

Usage

```
overlappedMode = slot[Z].smu[X].overlapped
slot[Z].smu[X].overlapped = overlappedMode
```

overlappedMode	Set the overlap mode: ▪ Disable: `slot[Z].smu[X].DISABLE` ▪ Enable: `slot[Z].smu[X].ENABLE`
Z	Module slot number
X	Module channel number

Details

When this attribute is set to disable, both sequential and measure overlapped commands proceed as in the order they are sent. For sequential commands, a command is processed fully before the next one is executed.

When this attribute is set to enable, a command returns as soon as it is able, even if it has not been reflected in the hardware. For example, the command for setting the source voltage level returns to Lua so it can process the next command, even if the instrument has not yet transitioned to the new voltage level.

Not all commands can be overlapped. If they can be, the command gets added to the processing queue of the channel and execute in order. Channels process their respective queue in parallel with other channels. If the command cannot be overlapped, the SMU inserts a `waitcomplete()` between commands. For example, if a `print(slot[Z].smu[X].source.rangev))` command is executed after a `slot[Z].smu[X].source.levelv = newLevel` command, the SMU completes the level change before printing the source range as if there were a `waitcomplete()` command executed between them.

This only applies to commands that specify settings. A command that retrieves a setting or a measure command waits for commands to complete before executing. Note that measure commands do not wait for commands to fully complete on a different channel.

This attribute is useful when performing operations on multiple channels. Changing the voltage level on channel 1 and channel 2 with this setting enabled should result in those changes occurring with less time between than if this setting was disabled.

Example

```
slot[1].smu[2].overlapped = slot[1].smu[2].ENABLE
```

Enables overlapped mode for channel 2 of the module installed in slot 1.

Also see

None

slot[Z].smu[X].reset()

This function turns off the output and resets the attributes that begin with `smu [X].` to their default settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].reset()`

Z	Module slot number
X	Module channel number

Details

This function turns off the output and returns the specified SMU to its default settings.

Example

```
slot[1].smu[1].reset()
```

Turns off the output and resets channel 1 of the module installed in slot 1 to its default settings.

Also see

None

slot[Z].smu[X].sense

This attribute contains the state of the sense mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].SENSE_2WIRE

Usage

```
senseMode = slot[Z].smu[X].sense
slot[Z].smu[X].sense = senseMode
```

senseMode	The sense setting: ▪ `slot[Z].smu[X].SENSE_2WIRE` or `slot[Z].smu[X].SENSE_LOCAL` ▪ `slot[Z].smu[X].SENSE_4WIRE` or `slot[Z].smu[X].SENSE_REMOTE` ▪ `slot[Z].smu[X].SENSE_AUTO` ▪ `slot[Z].smu[X].SENSE_CALA`
Z	Module slot number
X	Module channel number

Details

Source-measure operations are performed using either 2-wire local sense connections or 4-wire remote sense connections. When local or 2-wire sense is selected, the sense is configured for local sense operation. When remote or 4-wire sense is selected, the sense is configured for remote sense operation without the autosense circuit.

When `SENSE_AUTO` is selected, 4-wire sense also includes the autosense circuit. Autosense places resistors between the force and sense pairs to prevent unexpected voltages.

You can change the local or remote sense mode while the output is on.

The `slot[Z].smu[X].SENSE_CALA` mode is only used for calibration. You must enable calibration before selecting the calibration sense setting. The output must be off when this option is selected.

Example

```
slot[1].smu[1].sense = slot[1].smu[1].SENSE_REMOTE
```

Configures the sense mode to remote sense for channel 1 of the module installed in slot 1.

Also see

Comparison of 2-wire to 4-wire measurement techniques

slot[Z].smu[X].source.activelimitnY

This attribute contains the active negative source limit value presently in use by the module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Mainframe reset Module reset	Not applicable	Not applicable

Usage

activeValue = slot[*Z*].smu[*X*].source.activelimitn*Y*

activeValue	The active negative source limit
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

When an SMU module is configured to use asymmetric voltage or current source limits, the instrument will accept values outside of the source limit range. The starting source limit range is specified by the larger absolute value of the limits set by `slot[Z].smu[X].source.limitpY` and `slot[Z].smu[X].source.limitnY`. The source limit range is adjusted by the instrument as needed.

The lowest level that the module will limit when sourcing is 10% of the source limit range. If the smaller limit is less than that value, the module may limit the source to 10% of the source limit range.

For example, setting a positive current source limit of 1 A and a negative current source limit of -500 mA will result in the source limit range being set to 1 A and the source being limited to -500 mA, as they are on the same range. However, setting a positive current source limit of 1 A and a negative current source limit of -1 µA, may lead to the source being limited to 100 mA (10% of 1 A).

Tektronix suggests setting asymmetric limits and then using the `slot[Z].smu[X].source.activelimitpY` and `slot[Z].smu[X].source.activelimitnY` commands to monitor limit values during a test.

Example

```
slot[1].smu[1].source.limitpi = 100e-3
slot[1].smu[1].source.limitni = -100e-6
slot[1].smu[1].source.leveli = 50e-3
slot[1].smu[1].source.output = 1
print(slot[1].smu[1].source.limitni)
print(slot[1].smu[1].source.activelimitni)
```

Might return

```
-1e-6
-10e-3
```

This example sets the asymmetric positive and negative limits to 100 mA and -1 uA, respectively. The starting source limit range is automatically set to 100 mA. Sets the source level to 50 mA and turns the source output on. Printing the set negative source limit returns the user set value of -100 uA (-1e-6). Printing the active negative limit returns 10 % of the current source limit range, or -10 mA (-10e-3).

Also see

slot[Z].smu[X].source.activelimitpY (on page 509)

slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitpY (on page 516)

slot[Z].smu[X].source.activelimitpY

This attribute contains the actual positive source limit value presently in use by the module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Power cycle Mainframe reset Module reset	Not applicable	Not applicable

Usage

activeValue = slot[*Z*].smu[*X*].source.activelimitp*Y*

activeValue	The active positive source limit
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

When an SMU module is configured to use asymmetric voltage or current source limits, the instrument will accept values outside of the source limit range. The starting source limit range is specified by the larger absolute value of the limits set by `slot[Z].smu[X].source.limitpY` and `slot[Z].smu[X].source.limitnY`. The source limit range is adjusted by the instrument as needed.

The lowest level that the module will limit when sourcing is 10% of the source limit range. If the smaller limit is less than that value, the module may limit the source to 10% of the source limit range.

For example, setting a positive current source limit of 1 A and a negative current source limit of -500 mA will result in the source limit range being set to 1 A and the source being limited to -500 mA, as they are on the same range. However, setting a positive current source limit of 1 A and a negative current source limit of -1 µA, may lead to the source being limited to 100 mA (10% of 1 A).

Tektronix suggests setting asymmetric limits and then using the `slot[Z].smu[X].source.activelimitpY` and `slot[Z].smu[X].source.activelimitnY` commands to monitor limit values during a test.

Example

```
slot[1].smu[1].source.limitpi = 100e-3
slot[1].smu[1].source.limitni = -100e-6
slot[1].smu[1].source.leveli = 50e-3
slot[1].smu[1].source.output = 1
print(slot[1].smu[1].source.limitni)
print(slot[1].smu[1].source.activelimitni)
```

Might return

```
-1e-6
-10e-3
```

This example sets the asymmetric positive and negative limits to 100 mA and -1 uA, respectively. The starting source limit range is automatically set to 100 mA. Sets the source level to 50 mA and turns the source output on. Printing the set negative source limit returns the user set value of -100 uA (-1e-6). Printing the active negative limit returns 10% of the current source limit range, or -10 mA (-10e-3).

Also see

slot[Z].smu[X].source.activelimitnY (on page 508)
slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitpY (on page 516)

slot[Z].smu[X].source.atlimit

This attribute indicates if the source output is prevented from sourcing a voltage or current that is more than a set value.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (R)	Yes	Not applicable	Not saved	Not applicable

Usage

value = slot[*Z*].smu[*X*].source.atlimit

value	The state of the source limits: ▪ The limit function is in control of the source (source is limited): `true` ▪ The source function is in control of the output (source is not limited): `false`
Z	Module slot number
X	Module channel number

Details

This read-only attribute updates the status model with generated limit information. See the description of the Measurement Event Registers, Current Limit (ILMT), and Voltage Limit (VLMT).

Example

```
atlimit = slot[1].smu[2].source.atlimit
print(atlimit)
```

Reads and prints the source compliance state for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.levelY (on page 514)

slot[Z].smu[X].source.autorangeY

This attribute contains the state of the source autorange control (on or off).

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].ON

Usage

```
sourceAutorange = slot[Z].smu[X].source.autorangeY
slot[Z].smu[X].source.autorangeY = sourceAutorange
```

sourceAutorange	`slot[Z].smu[X].OFF`
	
	`slot[Z].smu[X].ON`

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

If autorange is enabled and you attempt to change the level of the selected source function while overlapped measurements are in progress, an error is generated and the setting is not applied.

When autorange is on, the source range value is changed to match the autorange value.

Example

```
slot[1].smu[2].source.autorangev = slot[1].smu[2].OFF
slot[1].smu[2].source.rangev = 20
```

Disables the autorange feature for channel 2 of the module installed in slot 1 and set a fixed source range of 20 V.

Also see

slot[Z].smu[X].measure.overlappedY() (on page 500)
slot[Z].smu[X].source.levelY (on page 514)
slot[Z].smu[X].source.lowrangeY (on page 518)
slot[Z].smu[X].source.rangeY (on page 528)

slot[Z].smu[X].source.delay

This attribute contains the additional delay that occurs when the source is changed.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	0

Usage

```
sDelay = slot[Z].smu[X].source.delay
slot[Z].smu[X].source.delay = sDelay
```

sDelay	The length of the delay in seconds: 0 to 300
Z	Module slot number
X	Module channel number

Details

This attribute allows for additional delay (settling time) after an output step. Set *sDelay* to a user-defined value (in seconds).

Example

```
slot[1].smu[1].source.delay = 3e-3
```

Sets a 3 ms delay after each source action while the source is turned on for channel 1 of the module installed in slot 1.

Also see

None

slot[Z].smu[X].source.func

This attribute configures the source function as either a voltage source or current source.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].FUNC_DC_VOLTAGE

Usage

```
sFunction = slot[Z].smu[X].source.func
slot[Z].smu[X].source.func = sFunction
```

sFunction	Set the source function: ▪ Current: `slot[Z].smu[X].FUNC_DC_CURRENT` ▪ Voltage: `slot[Z].smu[X].FUNC_DC_VOLTAGE`
Z	Module slot number
X	Module channel number

Details

This setting configures the source function as either voltage or current.

This command accepts the following options to ease 26xx migration of existing scripts. Setting source function with one of these options configures the source for the correct function. However, the return value will be one of the options noted for value under usage above.

Acceptable parameters for backwards compatibility with 26xx:

- `slot[Z].smu.FUNC_DCAMPS` for current
- `slot[Z].smu.FUNC_DCVOLTS` for voltage

Example

```
slot[1].smu[2].source.func = slot[1].smu[2].FUNC_DC_CURRENT
```

Configures the source to be DC current for channel 2 of the module installed in slot 1.

Also see

None

slot[Z].smu[X].source.levelY

This attribute configures the source level setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	Voltage: 0 Current: 0

Usage

```
sourceLevel = slot[Z].smu[X].source.levelY
slot[Z].smu[X].source.levelY = sourceLevel
```

sourceLevel	**MSMU60-2** ▪ **Voltage:** 0 V to 60 V ▪ **Current:** 0 A to 1.5 A **MSMU200-2** ▪ **Voltage:** 0 V to 200 V ▪ **Current:** 0 A to 1.5 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This attribute configures the output level of the voltage or current source.

The sign of *sourceLevel* dictates the polarity of the source. Positive values generate positive voltage or current from the high terminal of the source relative to the low terminal. Negative values generate negative voltage or current from the high terminal of the source relative to the low terminal.

For the MSMU200-2, levels above 1.5 A are pulsing-only. For more information, see the *MP5000 Series Source-Measure Unit Reference Manual*.

Example

```
slot[1].smu[2].source.levelv = 3.5
```

Sets the voltage source level of channel 2 of the SMU module installed in slot 1 to 3.5 V.

Also see

slot[Z].smu[X].source.atlimit (on page 510)
slot[Z].smu[X].source.func (on page 513)
slot[Z].smu[X].source.output (on page 524)

slot[Z].smu[X].source.limitnY

This attribute configures the negative source limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.limitY	Configuration list index	Voltage: −20 V Current: −100 mA

Usage

```
value = slot[Z].smu[X].source.limitnY
slot[Z].smu[X].source.limitnY = value
```

value	**MSMU60-2** ▪ **Voltage:** −20 mV to –60 V ▪ **Current:** –10 nA to –1.5 A **MSMU200-2** ▪ **Voltage:** −20 mV to –200 V ▪ **Current:** –10 nA to –1.5 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the negative source limit. The value must be negative.

Limits apply to DC values.

Example

```
slot[1].smu[2].source.limitnv = -2
```

Sets the negative voltage limit to -2 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.limitpY (on page 516)
slot[Z].smu[X].source.limitY (on page 517)
slot[Z].smu[X].source.pulse.limitnY (on page 525)

slot[Z].smu[X].source.limitpY

This attribute configures the positive source limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.limitY	Configuration list index	Voltage: 20 V Current: 100 mA

Usage

```
value = slot[Z].smu[X].source.limitpY
slot[Z].smu[X].source.limitpY = value
```

value	**MSMU60-2** ▪ **Voltage:** 0.02 V to 60 V ▪ **Current:** 1e−8 A to 1.5 A **MSMU200-2** ▪ **Voltage:** 0.02 V to 200 V ▪ **Current:** 1e−8 A to 1.5 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the positive source limit.

Limits apply to DC values.

Example

```
slot[1].smu[2].source.limitpv = 2
```

Sets the positive voltage limit to 2 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitY (on page 517)
slot[Z].smu[]X].source.pulse.limitpY (on page 526)

slot[Z].smu[X].source.limitY

This attribute symmetrically configures the positive and the negative source limit settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.limitnY slot[Z].smu[X].source.limitpY	Configuration list index	Voltage: 20 V Current: 100 mA

Usage

```
value = slot[Z].smu[X].source.limitY
slot[Z].smu[X].source.limitY = value
```

value	**MSMU60-2** ▪ **Voltage:** –0.02 V to 60 V ▪ **Current:** −10 nA to 1.5 A **MSMU200-2** ▪ **Voltage:** –0.02 V to 200 V ▪ **Current:** −10 nA to 1.5 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

The source limits prevent the instrument from sourcing a voltage or current over a set value.

Limits apply to DC values.

Example

```
slot[1].smu[2].source.limiti = 35e-3
```

Sets the positive current limit to 35 mA and the negative current limit to -35 mA for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitpY (on page 516)
slot[Z].smu[X].source.pulse.limitY (on page 527)

slot[Z].smu[X].source.lowrangeY

This attribute sets the lowest source range that is used when autorange is on.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	Voltage: 2 V Current: 100e−9 A (100 nA)

Usage

```
sourceRangeLow = slot[Z].smu[X].source.lowrangeY
slot[Z].smu[X].source.lowrangeY = sourceRangeLow
```

sourceRangeLow	Set the lowest source range lowest range to use when autorange is on: **MSMU60-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 60 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 1.5 A **MSMU200-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 200 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 7 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This attribute is used with source autorange to assign a lower limit on the automatically selected range. Lower ranges generally require greater settling times. By setting a low-range value that is higher than the automatically selected value, the SMU may be able to source small values with less settling time. If the instrument is set to autorange and it is on a range lower than the one specified by *sourceRangeLow*, the source range is changed to the range specified by *sourceRangeLow*.

The 7 A range on the MSMU200-2 is pulsing-only above 1.5 A. For more information, see the *MP5000 Series Source-Measure Unit Reference Manual*.

Example

```
slot[1].smu[1].source.lowrangei = 1e-6
```

Sets the current source low range attribute to 1 uA. This prevents the source from using lower ranges when sourcing current.

Also see

slot[Z].smu[X].source.autorangeY (on page 511)

slot[Z].smu[X].source.offfunc

This attribute sets the source function used (source 0 A or 0 V) when the output is turned off and the SMU is in normal output-off mode.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].FUNC_DC_VOLTAGE

Usage

```
offfunc = slot[Z].smu[X].source.offfunc
slot[Z].smu[X].source.offfunc = offfunc
```

offfunc	Source function: ▪ Current: `slot[Z].smu[X].FUNC_DC_CURRENT` ▪ Voltage: `slot[Z].smu[X].FUNC_DC_VOLTAGE`
Z	Module slot number
X	Module channel number

Details

This attribute controls the source function used when the output is turned off and `slot[Z].smu[X].source.offmode` is set to `slot[X].smu[Y].OFFMODE_NORMAL`. Set this attribute to voltage for the source to be a 0 V source when the output is off (`slot[Z].smu[X].offlimiti` is used). Set it to current for the source to be a 0 A source when the output is off (`slot[Z].smu[X].offlimitv` is used).

This attribute is only used when the `slot[Z].smu[X].source.offmode` attribute is set to `slot[Z].smu[X].OFFMODE_NORMAL`.

Example

```
slot[1].smu[1].source.offfunc = slot[1].smu[1].FUNC_DC_CURRENT
```

Sets the normal output-off mode to source 0 A when the output is turned off for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.offmode (on page 523)
slot[Z].smu[X].source.output (on page 524)

slot[Z].smu[X].source.offlimitnY

This attribute configures the negative source off limit setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.offlimitY	Configuration list index	Voltage: -60 V Current: -0.001 A (-1 mA)

Usage

```
value = slot[Z].smu[X].source.offlimitnY
slot[Z].smu[X].source.offlimitnY = value
```

value	**MSMU60-2** ▪ **Voltage:** 0 V to −60 V ▪ **Current:** 0 A to −100 mA **MSMU200-2** ▪ **Voltage:** 0 V to −200 V ▪ **Current:** 0 A to −100 mA
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

The off limit value must be negative.

Example

```
slot[1].smu[1].source.offlimitnv = -2
```

Sets the negative voltage off limit to −2 V for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.offlimitY (on page 522)

slot[Z].smu[X].source.offlimitpY

This attribute configures the positive source off limit setting.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.offlimitY	Configuration list index	Voltage: 60 V Current: 0.001 A (1 mA)

Usage

```
value = slot[Z].smu[X].source.offlimitpY
slot[Z].smu[X].source.offlimitpY = value
```

value	**MSMU60-2** ▪ **Voltage:** 0 V to 60 V ▪ **Current:** 0 A to 100 mA **MSMU200-2** ▪ **Voltage:** 0 V to 200 V ▪ **Current:** 0 A to 100 mA
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the positive source off limit setting.

Example

```
slot[1].smu[1].source.offlimitpv = 2
```

Sets the negative voltage off limit to 2 V for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.offlimitY (on page 522)

slot[Z].smu[X].source.offlimitY

This attribute configures the positive and negative source off limit settings symmetrically.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.offlimitnY slot[Z].smu[X].source.offlimitpY	Configuration list index	Voltage: 60 V Current: 0.001 A (1 mA)

Usage

value = slot[*Z*].smu[*X*].source.offlimit*Y*
slot[*Z*].smu[*X*].source.offlimit*Y* = value

value	**MSMU60-2** ▪ **Voltage:** 20 mV to 60 V ▪ **Current:** 10 nA to 100 mA **MSMU200-2** ▪ **Voltage:** 20 mV to 200 V ▪ **Current:** 10 nA to 100 mA
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures both the positive and negative source off limit settings symmetrically.

Example

```
slot[1].smu[1].source.offlimiti = 35e-3
```

Sets the positive current off limit to 35 mA and the negative current off limit to −35 mA.

Also see

slot[Z].smu[X].source.offlimitnY (on page 520)
slot[Z].smu[X].source.offlimitpY (on page 521)

slot[Z].smu[X].source.offmode

This attribute sets the mode when the source output is turned off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index	Configuration list index	slot[Z].smu[X].OFFMODE_NORMAL

Usage

```
sourceOffmode = slot[Z].smu[X].source.offmode
slot[Z].smu[X].source.offmode = sourceOffmode
```

sourceOffmode	The output-off mode: ▪ Normal: `slot[Z].smu[X].OFFMODE_NORMAL` ▪ High impedance: `slot[Z].smu[X].OFFMODE_HIGH_Z`
Z	Module slot number
X	Module channel number

Details

When this command is set to normal, the source function is configured to be either a 0 A current source or 0 V voltage source when the source is off, as set by the `slot[Z].smu[X].source.offfunc` attribute. The source off limit settings determine the limit settings.

When the mode is set to high impedance, the SMU opens the output relay when the output is turned off.

If you are using contact check, you cannot use the high impedance output setting.

Example

```
slot[1].smu[1].source.offmode = slot[1].smu[1].OFFMODE_HIGH_Z
```

Sets the output-off operation to open the output relay when the output is turned off for channel 1 of the module installed in slot 1.

Also see

slot[Z].smu[X].contact.check() (on page 484)
slot[Z].smu[X].source.offfunc (on page 519)
slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitpY (on page 516)
slot[Z].smu[X].source.limitY (on page 517)
slot[Z].smu[X].source.output (on page 524)

slot[Z].smu[X].source.output

This attribute enables or disables the source output.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset	Not saved	slot[Z].smu[X].OFF

Usage

```
value = slot[Z].smu[X].source.output
slot[Z].smu[X].source.output = value
```

value	The setting of the source output: ▪ `slot[Z].smu[X].OFF` ▪ `slot[Z].smu[X].ON` ▪ `slot[Z].smu[X].HIGH_Z`
Z	Module slot number
X	Module channel number

Details

When this attribute is read, it returns the output state of the source. When this attribute is set, it switches the output of the source on or off.

You can also select `slot[Z].smu[X].HIGH_Z` for this attribute. When the high-Z, or high impedance, output setting is selected and the output is turned off, the output relay opens, which disconnects external circuitry from the input and output of the SMU. To prevent excessive wear on the output relay, do not use the high-Z output setting for tests that turn the output off and on frequently. When the high-Z output setting is selected, the setting of `slot[Z].smu[X].source.offmode` is ignored.

If you are using contact check, you cannot use the high-Z output setting.

The source output setting is not stored in configuration lists.

Example

```
slot[1].smu[2].source.output = slot[1].smu[2].ON
```

Turns on the output for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].contact.check() (on page 484)
slot[Z].smu[X].source.offfunc (on page 519)
slot[Z].smu[X].source.offmode (on page 523)

slot[Z].smu[X].source.pulse.limitnY

This attribute configures the negative pulse limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.pulse.limitY	Configuration list index	Voltage: −20 V Current: −100 mA

Usage

```
value = slot[Z].smu[X].source.pulse.limitnY
slot[Z].smu[X].source.pulse.limitnY = value
```

value	**MSMU60-2** ▪ **Voltage:** −20 mV to –60 V ▪ **Current:** –10 nA to –1.5 A **MSMU200-2** ▪ **Voltage:** −20 mV to –200 V ▪ **Current:** –10 nA to –7 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the negative pulse limit. The value must be negative.

Example

```
slot[1].smu[2].source.pulse.limitnv = -2
```

Sets the negative voltage limit to -2 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.pulse.limitpY (on page 526)
slot[Z].smu[X].source.pulse.limitY (on page 527)

slot[Z].smu[X].source.pulse.limitpY

This attribute configures the positive pulse limit.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.pulse.limitY	Configuration list index	Voltage: 20 V Current: 100 mA

Usage

```
value = slot[Z].smu[X].source.pulse.limitpY
slot[Z].smu[X].source.pulse.limitpY = value
```

value	**MSMU60-2** ▪ **Voltage:** 0.02 V to 60 V ▪ **Current:** 1e−8 A to 1.5 A **MSMU200-2** ▪ **Voltage:** 0.02 V to 200 V ▪ **Current:** 1e−8 A to 7 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the positive pulse limit. The value must be positive.

Example

```
slot[1].smu[2].source.pulse.limitpv = 2
```

Sets the positive voltage limit to 2 V for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.pulse.limitnY (on page 525)
slot[Z].smu[X].source.pulse.limitY (on page 527)

slot[Z].smu[X].source.pulse.limitY

This attribute symmetrically configures the positive and negative pulse limit settings.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.pulse.limitnY slot[Z].smu[X].source.pulse.limitpY	Configuration list index	20 V 100 mA

Usage

```
value = slot[Z].smu[X].source.pulse.limitY
slot[Z].smu[X].source.pulse.limitY = value
```

value	**MSMU60-2** ▪ **Voltage:** –0.02 V to 60 V ▪ **Current:** −10 nA to 1.5 A **MSMU200-2** ▪ **Voltage:** –0.02 V to 200 V ▪ **Current:** −10 nA to 7 A
Z	Module slot number
X	Module channel number
Y	The source function; v = voltage, i = current

Details

The pulse limits prevent the instrument from pulsing a voltage or current over a set value.

Example

```
slot[1].smu[2].source.pulse.limiti = 35e-3
```

Sets the positive current limit to 35 mA and the negative current limit to -35 mA for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].source.pulse.limitnY (on page 525)
slot[Z].smu[X].source.pulse.limitpY (on page 526)

slot[Z].smu[X].source.rangeY

This attribute configures the source range setting to a fixed value for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Power cycle Mainframe reset Module reset Configuration list index slot[Z].smu[X].source.autorangeY	Configuration list index	Voltage: 2 V Current: 100 nA

Usage

```
value = slot[Z].smu[X].source.rangeY
slot[Z].smu[X].source.rangeY = value
```

value	Set to the maximum expected voltage or current: **MSMU60-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 60 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 1.5 A **MSMU200-2** ▪ **Voltage:** 200 mV, 2 V, 6 V, 20 V, 200 V ▪ **Current:** 100 nA, 1 µA, 10 µA, 100 µA, 1 mA, 10 mA, 100 mA, 1 A, 7 A
Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current

Details

This setting configures the source fixed range setting for the specified source function.

If autorange is on, the source range is set to the autorange value.

The range setting depends on the maximum voltage level setting (see `slot[Z].smu[X].measure.rangeY`). The 7 A range on the MSMU200-2 is pulsing-only above 1.5 A. For more information, see the *MP5000 Series Source-Measure Unit Reference Manual*.

Example

```
slot[1].smu[2].source.rangev = 12
```

Selects the 20 V range for channel 2 of the module installed in slot 1.

Also see

slot[Z].smu[X].measure.rangeY (on page 501)
slot[Z].smu[X].reset() (on page 506)
slot[Z].smu[X].source.autorangeY (on page 511)

slot[Z].smu[X].trigger.EVENT_AT_LIMIT

This constant contains the source limit event number of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

eventId = slot[*Z*].smu[*X*].trigger.EVENT_AT_LIMIT

eventId	The trigger event number
Z	Module slot number
X	Module channel number

Details

This constant contains the source limit event number of the trigger model.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.linearv(5, 5, 1)

-- Configure the current source limit.
slot[1].smu[1].source.limiti = 0.001
-- Configure the measurement.
READING_BUFFER = slot[1].smu[1].makebuffer(10)
slot[1].smu[1].trigger.measure.v(READING_BUFFER)
-- Create the trigger model.
slot[1].trigger.model.delete("myTM")
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.measure("myTM", "", 1, 1)
slot[1].trigger.model.addblock.branch.event("myTM", "", "abort", slot[1].smu[1].tr
igger.EVENT_AT_LIMIT)
slot[1].trigger.model.addblock.branch.always("myTM", "", "done")
slot[1].trigger.model.addblock.logevent("myTM", "abort", slot[1].trigger.model.LOG
_WARN_ABORT, "Abort myTM")
slot[1].trigger.model.addblock.nop("myTM", "done")
-- Turn on the output before the trigger model is initiated.
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model using a branch event block that waits for a limit event and executes the trigger model abort if the limit event is triggered. The setup includes a 1 kΩ resistor connected to the SMU, with a current limit set to 1 mA. This configuration restricts the voltage output to 1 V, placing the system in the limited state. The trigger model performs a source step to 5 V, which causes the system to exceed the limit and subsequently triggers the limit event when making a measurement

Also see

None

slot[Z].smu[X].trigger.measure.Y()

This function configures the measurements to be made during trigger model operation.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].trigger.measure.Y(readingBuffer)
slot[Z].smu[X].trigger.measure.iv(currentBuffer, voltageBuffer)
```

Z	Module slot number
X	Module channel number
Y	The measurement function; `v` = voltage, `i` = current, `r` = resistance, `p` = power
readingBuffer	The reading buffer; can be a user-defined or default reading buffer
currentBuffer	Reading buffer where current readings are stored; can be a user-defined buffer or a default reading buffer
voltageBuffer	Reading buffer where voltage readings are stored; can be a user-defined buffer or a default reading buffer

Details

This function configures the measurement subsystem for one or more synchronous measurements to be made while a trigger model is executed.

Measurements are in the following units of measure:

- `v` = volts
- `i` = amperes
- `r` = ohms
- `p` = watts

This command only configures the measure functions and reading buffers to be used during a trigger model; it does not cause measurements to be made. To make measurements, you must add a measure block to the trigger model using the `slot[Z].smu[X].trigger.model.addblock.measure()` command.

The `slot[Z].smu[X].measure.count` attribute is not used by the measure subsystem when a trigger model is executed. Instead, the measure count is passed in through the measure block command.

The reading buffers are cleared when `slot[Z].trigger.model.initiate()` is called unless the buffer is configured to append data. Reading buffers are not optional for this command.

The module only retains the last call to any one of these functions and only that measurement is made.

Example

```
--- Configure SMU channel 1 to measure voltage and save the result in the reading
buffer.
ReadingBuffer = slot[1].smu[1].makebuffer(10)
slot[1].smu[1].trigger.measure.v(ReadingBuffer)
--- Set the initial voltage output for SMU Ch1 to 2 V.
slot[1].smu[1].source.levelv = 2
--- Delete old trigger model instances and then create new one.
slot[1].trigger.model.delete("TM1")
slot[1].trigger.model.create("TM1")
--- Make the voltage measurement 10 times.
slot[1].trigger.model.addblock.source.output("TM1", "OutputOn", 1, 1)
slot[1].trigger.model.addblock.measure("TM1", "MeasureVoltage", 1, 10)
slot[1].trigger.model.addblock.source.output("TM1", "OutputOff", 1, 0)
--- Execute the trigger model.
slot[1].trigger.model.initiate("TM1")
waitcomplete()
-- Use printbuffer to return the 10 measurements.
printbuffer(1, ReadingBuffer.n, ReadingBuffer)
```

A reading buffer is configured for voltage measurements and a measure block captures measurements for the module installed in slot 1.

Also see

slot[Z].trigger.model.addblock.measure() (on page 590)
slot[Z].trigger.model.addblock.measureoverlapped() (on page 592)

slot[Z].smu[X].trigger.source.linearY()

This function configures a linear sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].trigger.source.linearY(start, end, points)`

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current
start	Start source level
end	End source level
points	Number of points used to calculate the step size

Details

This command sets up a linear sweep by configuring the start source level, end source level, and the number of steps to increase or decrease from the starting source level to the ending source level. When you use a linear staircase sweep, the voltage or current source increases or decreases in fixed steps. Each point is equally-spaced between the start and stop.

The *points* parameter does not set the number of steps in a sweep. Instead, it is used to generate a list of source values to be used by a subsequent sweep.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The SMU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

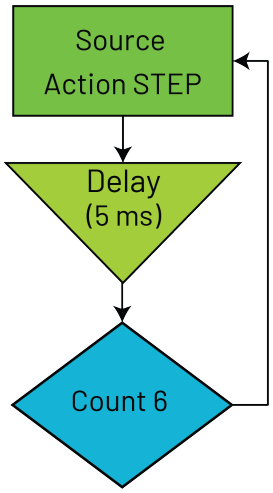
- `slot[Z].smu[X].trigger.source.linearY()`
- `slot[Z].smu[X].trigger.source.listY()`
- `slot[Z].smu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].smu[1].trigger.source.linearv(1, 5, 6)
```

Configure a six-point linear sweep from 1 V to 5 V on channel 1 in slot 1.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

slot[Z].smu[X].trigger.source.listY()

This function configures a list sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].smu[X].trigger.source.listY(SourceLevels)`

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current
SourceLevels	List of source levels; see **Details**

Details

This function configures a sweep from a predetermined list passed through the *sourceLevels* parameter. The number of elements in *sourceLevels* does not set the number of steps in a sweep. Instead, it provides the list of source values to be used by a subsequent sweep. An example of a *sourceLevels* list is `{1, 2, 3}`.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The SMU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

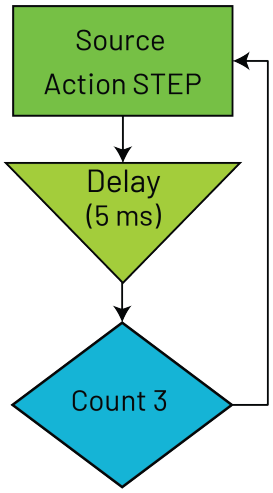
- `slot[Z].smu[X].trigger.source.linearY()`
- `slot[Z].smu[X].trigger.source.listY()`
- `slot[Z].smu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].smu[1].trigger.source.listv({1, 2, 3})
```

Configure a list sweep using specific source levels 1 V, 2 V, and 3 V on channel 1 in slot 1

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step (on page 610)

slot[Z].smu[X].trigger.source.logY()

This function configures a logarithmic sweep for the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].smu[X].trigger.source.logY(start, end, points)
slot[Z].smu[X].trigger.source.logY(start, end, points, asymptote)
```

Z	Module slot number
X	Module channel number
Y	The source function; `v` = voltage, `i` = current
start	Start source level
end	End source level
points	Number of points used to calculate the step size
asymptote	The asymptotic offset value (optional)

Details

This command sets up a logarithmic sweep by configuring the start source level, end source level, and the number of points to increase or decrease from the starting source level to the ending source level.

The points parameter does not set the number of steps in a sweep. Instead, it generates a list of source values to be used by a subsequent sweep.

A logarithmic staircase sweep is similar to a linear staircase sweep. In a logarithmic sweep, however, the steps are scaled logarithmically.

The *asymptote* parameter customizes the inflection and offset of the source value curve. This allows log sweeps to cross zero. Setting this parameter to zero provides a conventional logarithmic sweep. The *asymptote* value is the value that the curve has at either positive or negative infinity, depending on the direction of the sweep.

The *asymptote* value must be outside the range defined by the start and end values. It must not be equal to or between the start and end values.

To advance through the configured sweep, use source action blocks in a trigger model. Available blocks include `slot[Z].trigger.model.addblock.source.action.step`, `.set`, `.skip`, and `.bias`. When you use trigger model source action blocks to progress through the sweep:

- If the number of source block executions exceeds the configured number of levels, the sweep wraps to the beginning.
- If the number of executions is less than the configured number of levels, the remaining source values are ignored and not used.

The SMU stores the most recent configured source action. The last call to a sweep command is used for the source action in a trigger model. The sweep commands are:

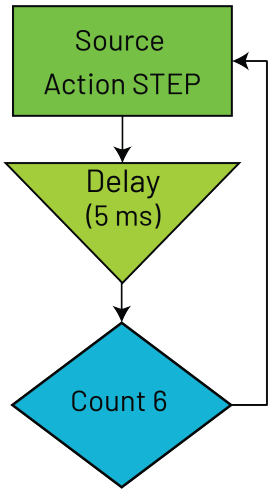
- `slot[Z].smu[X].trigger.source.linearY()`
- `slot[Z].smu[X].trigger.source.listY()`
- `slot[Z].smu[X].trigger.source.logY()`

Example

```
--- Configure the sweep. ---
slot[1].smu[1].trigger.source.logv(1, 5, 6)
```

Configure a six-point logarithmic sweep from 1 V to 5 V on channel 1 in slot 1.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.set() (on page 606)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

slot[Z].smu[X].waitcomplete()

This function waits for all previously started overlapped commands to complete on the specified channel of a module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	No			

Usage

`slot[Z].smu[X].waitcomplete()`

Z	Module slot number
X	Module channel number

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `slot[Z].smu[X].waitcomplete()` command is a sequential command that can be used as a synchronization point between channels. It does not return until all previously started overlapped commands on the specified channel of a module are complete. This effectively suspends the execution of all commands following `slot[Z].smu[X].waitcomplete()` until then.

Example

```
slot[1].smu[1].waitcomplete()
```

Suspends execution of commands on channel 1 of the module installed in slot 1.

Also see

None

slot[Z].status.*

This attribute contains the registers of the status register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—-	—	—	—
.condition (R)	Yes	Not applicable	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	137 (all bits set)

Usage

```
measurementRegister = slot[Z].status.condition
measurementRegister = slot[Z].status.enable
measurementRegister = slot[Z].status.event
measurementRegister = slot[Z].status.ntr
measurementRegister = slot[Z].status.ptr
slot[Z].status.enable = measurementRegister
slot[Z].status.ntr = measurementRegister
slot[Z].status.ptr = measurementRegister
```

measurementRegister	The status of the slot status summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the slot status summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 129 is returned for the enable register. The binary equivalent is `0000 0000 1000 0001`. This value indicates that bits B0 (MSB) and B7 (OSB) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	`slot[Z].status.MSB` Set bit is a summary of the `slot[Z].status.measurement` register. Bit B0 decimal value: 1
B1 to B2	Not used

Bit	Value
B3	`slot[Z].status.QSB` Set bit is a summary of the `slot[Z].status.questionable` register. Bit B3 decimal value: 8
B4 to B6	Not used.
B7	`slot[Z].status.OSB` Set bit is a summary of the `slot[Z].status.operation` register. Bit B7 decimal value: 128
B8 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B0 and B7, set *measurementRegister* to 129 (which is the sum of 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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
slot[1].status.enable = slot[1].status.MSB
```

Sets the MSB bit of the status summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.*

This attribute contains the measurement event register set.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	6,275 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.condition
measurementRegister = slot[Z].status.measurement.enable
measurementRegister = slot[Z].status.measurement.event
measurementRegister = slot[Z].status.measurement.ntr
measurementRegister = slot[Z].status.measurement.ptr
slot[Z].status.measurement.enable = measurementRegister
slot[Z].status.measurement.ntr = measurementRegister
slot[Z].status.measurement.ptr = measurementRegister
```

measurementRegister	The status of the measurement event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate various bit settings
Z	Module slot number

Details

These attributes read or write the measurement event registers.

Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume value 130 is returned for the enable register. The binary equivalent is 0000 0000 1000 0001. This value indicates that bit B0 (VLMT) and bit B7 (ROF) are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	`slot[Z].status.measurement.VLMT` Set bit is a summary of the `slot[Z].status.measurement.voltage_limit` register. Bit B0 decimal value: 1

Bit	Value
B1	`slot[Z].status.measurement.ILMT` Set bit is a summary of the `slot[Z].status.measurement.current_limit` register. Bit B1 decimal value: 2
B2 to B6	Not used
B7	`slot[Z].status.measurement.ROF` Set bit is a summary of the `slot[Z].status.measurement.reading_overflow` register. Bit B7 decimal value: 128
B8 to B10	Not used
B11	`slot[Z].status.measurement.INT` Set bit indicates that interlock has been asserted. Bit B11 decimal value: 2,048
B12	`slot[Z].status.measurement.INST` Set bit indicates that a bit in the measurement instrument summary register is set. Bit B12 decimal value: 4,096
B13 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B7, set *measurementRegister* to 130 (which is the sum of 2 + 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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
slot[1].status.measurement.enable = slot[1].status.measurement.ILMT
```

For slot 1, sets the current limit (B1) bit of the Measurement Event enable register.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.current_limit.*

This attribute contains the measurement event current limit summary registers.

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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.current_limit.condition
measurementRegister = slot[Z].status.measurement.current_limit.enable
measurementRegister = slot[Z].status.measurement.current_limit.event
measurementRegister = slot[Z].status.measurement.current_limit.ntr
measurementRegister = slot[Z].status.measurement.current_limit.ptr
slot[Z].status.measurement.current_limit.enable = measurementRegister
slot[Z].status.measurement.current_limit.ntr = measurementRegister
slot[Z].status.measurement.current_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement event current limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the measurement event current limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 and bit B2 are set.

Bits B1 and B2 report the last status of the source while the output was turned on.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.current_limit.SMU1` Set bit indicates that the SMU 1 current limit was exceeded. Bit B1 decimal value: 2 Binary value: 0000 0010

Bit	Value
B2	`slot[Z].status.measurement.current_limit.SMU2` Set bit indicates that the SMU 2 current limit was exceeded. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.measurement.current_limit.enable = slot[1].status.measurement.current_limit.SMU1
```

Sets the SMU1 bit of the Measurement Event Current Limit Summary enable register.

Also see

Measurement event registers (on page 661)
slot[Z].status.measurement.instrument.smu[X].* (on page 544)

slot[Z].status.measurement.instrument.*

This attribute contains the registers of the measurement event instrument summary 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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.instrument.condition
measurementRegister = slot[Z].status.measurement.instrument.enable
measurementRegister = slot[Z].status.measurement.instrument.event
measurementRegister = slot[Z].status.measurement.instrument.ntr
measurementRegister = slot[Z].status.measurement.instrument.ptr
slot[Z].status.measurement.instrument.enable = measurementRegister
slot[Z].status.measurement.instrument.ntr = measurementRegister
slot[Z].status.measurement.instrument.ptr = measurementRegister
```

measurementRegister	The status of the measurement event instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the measurement event instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, assume the value 6 is returned for the enable register. The binary equivalent is 0000 0000 0000 0110. This value indicates that bit B1 (SMU1) and bit B2 (SMU2) are set.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.instrument.SMU1` Set bit indicates one or more enabled bits of the measurement event SMU 1 summary register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.measurement.instrument.SMU2` Set bit indicates one or more enabled bits of the measurement event SMU 2 summary register is set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.measurement.instrument.enable = slot[1].status.measurement.instrume
nt.SMU1
```

Sets the SMU 1 bit of the measurement event instrument summary enable register for the instrument in slot 1 using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.instrument.smu[X].*

This attribute contains the registers of the measurement event SMU 1 summary 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	131 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.instrument.smu[X].condition
measurementRegister = slot[Z].status.measurement.instrument.smu[X].enable
measurementRegister = slot[Z].status.measurement.instrument.smu[X].event
measurementRegister = slot[Z].status.measurement.instrument.smu[X].ntr
measurementRegister = slot[Z].status.measurement.instrument.smu[X].ptr
slot[Z].status.measurement.instrument.smu[X].enable = measurementRegister
slot[Z].status.measurement.instrument.smu[X].ntr = measurementRegister
slot[Z].status.measurement.instrument.smu[X].ptr = measurementRegister
```

measurementRegister	The status of the instrument measurement status SMU *Z* summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number
X	Channel of the SMU: 1 or 2

Details

These attributes are used to read or write to the measurement event SMU summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value

B0	`slot[Z].status.measurement.instrument.smu[X].VLMT`
	`slot[Z].status.measurement.instrument.smu[X].VOLTAGE_LIMIT`
	Set bit indicates that the voltage limit was exceeded.
	Bit B0 decimal value: 1
	This bit is updated only when a measurement is made or `slot[Z].smu[X].source.atlimit` is invoked.

B1	`slot[Z].status.measurement.instrument.smu[X].ILMT`
	`slot[Z].status.measurement.instrument.smu[X].CURRENT_LIMIT`
	Set bit indicates that the current limit was exceeded.
	Bit B1 decimal value: 2
	This bit is updated only when a measurement is made or `slot[Z].smu[X].source.atlimit` is invoked.

B2 to B6	Not used

B7	`slot[Z].status.measurement.instrument.smu[X].ROF`
	`slot[Z].status.measurement.instrument.smu[X].READING_OVERFLOW`
	Set bit indicates that an overflow reading has been detected.
	Bit B7 decimal value: 128

B8 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B8, set *measurementRegister* to 130 (which is the sum of 2 + 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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example

```
slot[2].status.measurement.instrument.smu[1].enable = slot[2].status.measurement.instrument.smu[1].VLMT
```

For slot 2, sets the VLMT bit of the Measurement Event SMU channel 1 Summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.reading_overflow.*

This attribute contains the measurement event reading overflow summary 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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.reading_overflow.condition
measurementRegister = slot[Z].status.measurement.reading_overflow.enable
measurementRegister = slot[Z].status.measurement.reading_overflow.event
measurementRegister = slot[Z].status.measurement.reading_overflow.ntr
measurementRegister = slot[Z].status.measurement.reading_overflow.ptr
slot[Z].status.measurement.reading_overflow.enable = measurementRegister
slot[Z].status.measurement.reading_overflow.ntr = measurementRegister
slot[Z].status.measurement.reading_overflow.ptr = measurementRegister
```

measurementRegister	The status of the measurement reading overflow summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings
Z	Module slot number

Details

These attributes are used to read or write to the measurement event reading overflow summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, assume the value 2 is returned for the enable register. The binary equivalent is 0000 0000 0000 0010. This value indicates that bit B1 is set.

Bits B1 and B2 report the last measurement that was made. If a reading overflow occurs, the bit is set. The bit is cleared when a non-overflowed reading is made.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.reading_overflow.SMU1` Set bit indicates that an overflow reading has been detected for SMU 1. Bit B1 decimal value: 2 Binary value: 0000 0010

Bit	Value
B2	`slot[Z].status.measurement.reading_overflow.SMU2` Set bit indicates that an overflow reading has been detected for SMU 2. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to constants, *measurementRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.measurement.reading_overflow.enable = slot[1].status.measurement.reading_overflow.SMU1
```

Sets the SMU 1 bit of the measurement reading overflow summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.measurement.voltage_limit.*

This attribute contains the measurement event voltage limit summary 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	6 (all bits set)

Usage

```
measurementRegister = slot[Z].status.measurement.voltage_limit.condition
measurementRegister = slot[Z].status.measurement.voltage_limit.enable
measurementRegister = slot[Z].status.measurement.voltage_limit.event
measurementRegister = slot[Z].status.measurement.voltage_limit.ntr
measurementRegister = slot[Z].status.measurement.voltage_limit.ptr
slot[Z].status.measurement.voltage_limit.enable = measurementRegister
slot[Z].status.measurement.voltage_limit.ntr = measurementRegister
slot[Z].status.measurement.voltage_limit.ptr = measurementRegister
```

measurementRegister	The status of the measurement voltage limit summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other decimal values indicate bit settings
Z	Module slot number

Details

These attributes read or write to the measurement event voltage limit summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bits B1 and B2 report the last status of the source while the output was turned on. If the limit is reached when the output is off, it is not reset until the source is turned on and is within limits.

Bit	Value
B0	Not used
B1	`slot[Z].status.measurement.voltage_limit.SMU1` Set bit indicates the enabled VLMT bit for the SMU 1 measurement register is set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.measurement.voltage_limit.SMU2` Set bit indicates the enabled VLMT bit for the SMU 2 measurement register is set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *measurementRegister* can be set to the decimal equivalent of the bit to set. To set more than one bit of the register, set *measurementRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *measurementRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[2].status.measurement.voltage_limit.enable = slot[2].status.measurement.voltage_limit.SMU2
```

Sets the SMU2 bit of the Measurement Event Voltage Limit Summary enable register using a constant.

Also see

Measurement event registers (on page 661)

slot[Z].status.operation

This attribute contains the operation status SMU summary 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	8217 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.condition
operationRegister = slot[Z].status.operation.enable
operationRegister = slot[Z].status.operation.event
operationRegister = slot[Z].status.operation.ntr
operationRegister = slot[Z].status.operation.ptr
slot[Z].status.operation.enable = operationRegister
slot[Z].status.operation.ntr = operationRegister
slot[Z].status.operation.ptr = operationRegister
```

operationRegister	The status of the Operation Status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Operation Status register. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `24` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 1000. This value indicates that bit B3 and bit B4 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	0	0	0	1	1	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	`slot[Z].status.operation.CAL` Set bit indicates that SMU is unlocked for calibration. Bit B0 decimal value: 1
B1 to B2	Not used

Bit	Value and description
B3	`slot[Z].status.operation.SWE` Set bit indicates that SMU is sweeping. Bit B3 decimal value: 8
B4	`slot[Z].status.operation.MEAS` Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B4 decimal value: 16
B5 to B6	Not used
B7	`slot[Z].status.operation.PLE` Set bit indicates that SMU pulse limits have been exceeded. Bit B7 decimal value: 128
B8 to B12	Not used
B13	`slot[Z].status.operation.INST` An enabled bit in the Operation Status Instrument Summary event register is set. Bit B13 decimal value: 8,192
B14 to B14	Not used

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B0 and B4, set *operationRegister* to 17 (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)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
slot[1].status.operation.enable = slot[1].status.operation.MEAS
```

Use a constant to set bit B4 of the Operation Status register for slot 1.

Example 2

```
slot[1].status.operation.enable = 24
```

Use the decimal value to set bits B3 and B4 of the Operation Status register for slot 1.

Also see

SMU Operation Status Registers (on page 658)

slot[Z].status.operation.calibrating.*

This attribute contains the operation status calibration summary 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	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.calibrating.condition
operationRegister = slot[Z].status.operation.calibrating.enable
operationRegister = slot[Z].status.operation.calibrating.event
operationRegister = slot[Z].status.operation.calibrating.ntr
operationRegister = slot[Z].status.operation.calibrating.ptr
slot[Z].status.operation.calibrating.enable = operationRegister
slot[Z].status.operation.calibrating.ntr = operationRegister
slot[Z].status.operation.calibrating.ptr = operationRegister
```

operationRegister	The status of the operation calibrating event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Operation Status Calibration Summary register. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.operation.calibrating.SMU1` Set bit indicates that SMU1 is unlocked for calibration. Bit B1 decimal value: 2 Binary value: 0000 0010

Bit	Value
B2	`slot[Z].status.operation.calibrating.SMU2` Set bit indicates that SMU2 is unlocked for calibration. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

Example

```
slot[1].status.operation.calibrating.enable = slot[1].status.operation.calibrating
.SMU1
```

Uses a constant to set the SMU1 bit of the operation status calibration summary enable register for slot 1.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.instrument.*

This attribute contains the Operation Status Instrument Summary 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	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.instrument.condition
operationRegister = slot[Z].status.operation.instrument.enable
operationRegister = slot[Z].status.operation.instrument.event
operationRegister = slot[Z].status.operation.instrument.ntr
operationRegister = slot[Z].status.operation.instrument.ptr
slot[Z].status.operation.instrument.enable = operationRegister
slot[Z].status.operation.instrument.ntr = operationRegister
slot[Z].status.operation.instrument.ptr = operationRegister
```

operationRegister	The status of the operation event register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set

Details

These attributes are used to read or write to the operation status instrument summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`slot[Z].status.operation.instrument.SMU1`	Set bit indicates one or more enabled bits for the operation status SMU 1 summary register are set. Bit B1 decimal value: 2
B2	`slot[Z].status.operation.instrument.SMU2`	Set bit indicates one or more enabled bits for the operation status SMU 2 summary register are set. Bit B2 decimal value: 4
B3 to B15	Not used	Not applicable.

As an example, to set bit B1 of the operation status instrument summary enable register, set `slot[1].status.operation.instrument.enable = slot[1].status.operation.instrument.SMU1`

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B2, set *`operationRegister`* to 6 (the sum of 2 + 4).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
operationRegister = slot[1].status.operation.instrument.SMU1 +
   slot[1].status.operation.instrument.SMU2
slot[1].status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B2 of the Operation Status Instrument Summary enable register using constants.

Example 2

```
-- 6 = binary 0000 0000 0000 0110
operationRegister = 6
slot[1].status.operation.instrument.enable = operationRegister
```

Sets bit B1 and bit B2 of the Operation Status Instrument Summary enable register using a decimal value.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.instrument.smu[X].*

This attribute contains the Operation Status SMU[*X*] Summary 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	25 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.instrument.smu[X].condition
operationRegister = slot[Z].status.operation.instrument.smu[X].enable
operationRegister = slot[Z].status.operation.instrument.smu[X].event
operationRegister = slot[Z].status.operation.instrument.smu[X].ntr
operationRegister = slot[Z].status.operation.instrument.smu[X].ptr
slot[Z].status.operation.instrument.smu[X].enable = operationRegister
slot[Z].status.operation.instrument.smu[X].ntr = operationRegister
slot[Z].status.operation.instrument.smu[X].ptr = operationRegister
```

Z	Module slot number
operationRegister	The status of the operation status SMU[*X*]; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
X	Source-measure unit (SMU) channel (for example, `slot[2]status.operation.instrument.smu[1].enable` applies to the SMU channel 1 in slot 2)

Details

These attributes are used to read or write to the Operation Status SMU[X] Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of `24` is read as the value of the condition register, the binary equivalent is 0000 0000 0001 1000. This value indicates that bit B3 and bit B4 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	0	0	0	1	1	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	`slot[Z].status.operation.instrument.smu[X].CAL` Set bit indicates that the selected SMU calibration is unlocked. Bit B0 decimal value: 1
B1 to B2	Not used
B3	`slot[Z].status.operation.instrument.smu[X].SWE` Set bit indicates that the selected SMU is sweeping. Bit B3 decimal value: 8
B4	`slot[Z].status.operation.instrument.smu[X].MEAS` Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement. Bit B4 decimal value: 16
B5 to B6	Not used
B7	`slot[Z].status.operation.instrument.smu[X].PLE` Set bit indicates that the SMU pulse limits have been exceeded. Bit B7 decimal value: 128
B8 to B15	Not used

As an example, to set bit B0 of the operation status SMU 1 summary enable register, set `slot[1].status.operation.instrument.smu[1].enable = slot[1].status.operation.instrument.smu[1].CAL`.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights.

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
slot[1].status.operation.instrument.smu[1].enable = slot[1].status.operation.instr
ument.smu[1].MEAS
```

Use a constant to set bit B4 of the Operation Status Summary enable register for SMU channel 1 in slot 1.

Example 2

```
slot[2].status.operation.instrument.smu[1].enable = 24
```

Use the decimal value to set bits B3 and B4 of the Operation Status Summary enable register of SMU channel 1 in slot 2.

Also see

Operation Status Registers (on page 658)

slot[Z].status.operation.measuring.*

This attribute contains the Operation Status Measuring Summary 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	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.measuring.condition
operationRegister = slot[Z].status.operation.measuring.enable
operationRegister = slot[Z].status.operation.measuring.event
operationRegister = slot[Z].status.operation.measuring.ntr
operationRegister = slot[Z].status.operation.measuring.ptr
slot[Z].status.operation.measuring.enable = operationRegister
slot[Z].status.operation.measuring.ntr = operationRegister
slot[Z].status.operation.measuring.ptr = operationRegister
```

operationRegister	The status of the operation status measuring summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bit settings

Details

These attributes are used to read or write to the operation status measuring summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.

Bit	Value	Description
B1	`slot[Z].status.operation.measuring.SMU1`	Bit is set when SMU 1 is making an overlapped measurement. It is not set when making a normal synchronous measurement. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.operation.measuring.SMU2`	Bit is set when SMU 2 is making an overlapped measurement. It is not set when making a normal synchronous measurement. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *operationRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.operation.measuring.enable = status.operation.measuring.SMU1
```

Uses a constant to set the SMU 1 bit of the Operation Status Measuring Summary enable register.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.sweeping.*

This attribute contains the Operation Status Sweeping Summary 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	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.sweeping.condition
operationRegister = slot[Z].status.operation.sweeping.enable
operationRegister = slot[Z].status.operation.sweeping.event
operationRegister = slot[Z].status.operation.sweeping.ntr
operationRegister = slot[Z].status.operation.sweeping.ptr
slot[Z].status.operation.sweeping.enable = operationRegister
slot[Z].status.operation.sweeping.ntr = operationRegister
slot[Z].status.operation.sweeping.ptr = operationRegister
```

operationRegister	The status of the operation status sweeping summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Operation Status Sweeping Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`slot[Z].status.operation.sweeping.SMU1`	Set bit indicates that SMU 1 has a source action trigger block in the trigger model when the trigger model is initiated. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.operation.sweeping.SMU2`	Set bit indicates SMU 2 has a source action trigger block in the trigger model when the trigger model is initiated. Bit B2 decimal value: 4 Binary value: 0000 0100

Bit	Value	Description
B3 to B15	Not used	Not applicable.

In addition to the above constants, *operationRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *operationRegister* to 6 (the sum of 2 + 4).

Example

```
slot[1].status.operation.sweeping.enable = slot[1].status.operation.sweeping.SMU1
```

Uses a constant to set the SMU 1 bit of the Operation Status Sweeping Summary enable register for slot 1.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.operation.pulselimitsexceeded

This attribute contains the Operation Status Pulse Limits Exceeded Summary 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	6 (all bits set)

Usage

```
operationRegister = slot[Z].status.operation.pulselimitsexceeded.condition
operationRegister = slot[Z].status.operation.pulselimitsexceeded.enable
operationRegister = slot[Z].status.operation.pulselimitsexceeded.event
operationRegister = slot[Z].status.operation.pulselimitsexceeded.ntr
operationRegister = slot[Z].status.operation.pulselimitsexceeded.ptr
slot[Z].status.operation.pulselimitsexceeded.enable = operationRegister
slot[Z].status.operation.pulselimitsexceeded.ntr = operationRegister
slot[Z].status.operation.pulselimitsexceeded.ptr = operationRegister
```

operationRegister	The status of the operation status pulse limits exceeded summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Operation Status Pulse Limits Exceeded Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0	Not used	Not applicable.
B1	`slot[Z].status.operation.pulselimitsexceeded.SMU1`	Set bit indicates that one or more enabled bits for the operation pulse limits exceeded status SMU 1 summary register are set. Binary value: 0000 0010
B2	`slot[Z].status.operation.pulselimitsexceeded.SMU2`	Set bit indicates that one or more enabled bits for the operation pulse limits exceeded status SMU 2 summary register are set. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used	Not applicable.

In addition to the above constants, *`operationRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`operationRegister`* to the sum of their decimal weights. For example, to set bits B1 and B2, set *`operationRegister`* to 6 (the sum of 2 + 4).

Example

```
slot[1].status.operation.pulselimitsexceeded.enable = slot[1].status.operation.pulselimitsexceeded.SMU1
```

Uses a constant to set the SMU 1 bit of the peration status pulse limits exceeded enable register for slot 1.

Also see

Operation Status Registers (on page 658)
status.operation.* (on page 244)

slot[Z].status.questionable.*

These attributes manage the questionable status register set of the status model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute	—	—	—	—
.condition (R)	Yes	Not applicable	Not saved	Not applicable
.enable (RW)	Yes	Status reset	Not saved	0
.event (R)	Yes	Status reset	Not saved	0
.ntr (RW)	Yes	Status reset	Not saved	0
.ptr (RW)	Yes	Status reset	Not saved	12,544 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.condition
questionableRegister = slot[Z].status.questionable.enable
questionableRegister = slot[Z].status.questionable.event
questionableRegister = slot[Z].status.questionable.ntr
questionableRegister = slot[Z].status.questionable.ptr
slot[Z].status.questionable.enable = questionableRegister
slot[Z].status.questionable.ntr = questionableRegister
slot[Z].status.questionable.ptr = questionableRegister
```

questionableRegister	The status of the questionable status register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the questionable status registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For example, if a value of 4,352 is read as the value of the condition register, the binary equivalent is 0001 0001 0000 0000. This value indicates that bits B8 and B12 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value	Description
B0 to B7	Not used	Not available

Bit	Value	Description
B8	`slot[Z].status.questionable.CAL`	An enabled bit in the questionable status calibration summary event register is set. Bit B8 decimal value: 256
B9	`slot[Z].status.questionable.WARMUP`	An enabled bit in the questionable status warmup summary event register is set. Bit B9 decimal value: 512
B10	`slot[Z].status.questionable.ACAL`	An enabled bit in the questionable status automatic calibration expired summary event register is set. Bit B10 decimal value: 1,024
B11	Not used	Not available
B12	`slot[Z].status.questionable.OTEMP`	An enabled bit in the questionable status over temperature summary event register is set. Bit B12 decimal value: 4,096
B13	`slot[Z].status.questionable.INST`	An enabled bit in the questionable status instrument summary event register is set. Bit B13 decimal value: 8,192
B14 to B15	Not used	Not available

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *questionableRegister* to the sum of their decimal weights. For example, to set bits B12 and B13, set *questionableRegister* to 12,288 (which is the sum of 4,096 + 8,192).

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

Example 1

```
slot[1].status.questionable.enable = slot[1].status.questionable.OTEMP
```

Uses a constant to set the OTEMP bit of the questionable status enable register for slot 1.

Example 2

```
slot[1].status.questionable.enable = slot[1].status.questionable.CAL
status.questionable.enable = 256
```

Both of these commands set the CAL enable bit (B8).

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)

slot[Z].status.questionable.calibration.*

This attribute contains the questionable status calibration summary 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	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.calibration.condition
questionableRegister = slot[Z].status.questionable.calibration.enable
questionableRegister = slot[Z].status.questionable.calibration.event
questionableRegister = slot[Z].status.questionable.calibration.ntr
questionableRegister = slot[Z].status.questionable.calibration.ptr
slot[Z].status.questionable.calibration.enable = questionableRegister
slot[Z].status.questionable.calibration.ntr = questionableRegister
slot[Z].status.questionable.calibration.ptr = questionableRegister
```

questionableRegister	The status of the questionable status calibration summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Questionable Status Calibration Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0	Not used

Bit	Value and description
B1	`slot[Z].status.questionable.calibration.SMU1` Set bit indicates that the SMU 1 calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.questionable.calibration.SMU2` Set bit indicates that the SMU 2 calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *questionableRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.questionable.calibration.enable = slot[1].status.questionable.calibration.SMU1
```

Uses a constant to set the SMU1 bit of the questionable status calibration summary enable register for slot 1.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.questionable.* (on page 434)

slot[Z].status.questionable.instrument.*

This attribute contains the questionable status instrument summary 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	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.instrument.condition
questionableRegister = slot[Z].status.questionable.instrument.enable
questionableRegister = slot[Z].status.questionable.instrument.event
questionableRegister = slot[Z].status.questionable.instrument.ntr
questionableRegister = slot[Z].status.questionable.instrument.ptr
slot[Z].status.questionable.instrument.enable = questionableRegister
slot[Z].status.questionable.instrument.ntr = questionableRegister
slot[Z].status.questionable.instrument.ptr = questionableRegister
```

questionableRegister	The status of the questionable status instrument summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the Questionable Status Instrument Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.questionable.instrument.SMU1` Set bit indicates one or more enabled bits for the SMU 1 questionable register are set. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.questionable.instrument.SMU2` Set bit indicates one or more enabled bits for the SMU 2 questionable register are set. Bit B2 decimal value: 4 Binary value: 0000 0100

Bit	Value
B3 to B15	Not used

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (which is the sum of 2 + 4).

Example

```
slot[1].status.questionable.instrument.enable = slot[1].status.questionable.instrument.SMU1
```

Uses a constant to set the SMU1 bit of the questionable status instrument summary enable register for slot 1.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.questionable.* (on page 434)

slot[Z].status.questionable.instrument.smu[X].*

This attribute contains the questionable status SMU summary 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	4352 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.instrument.smu[X].condition
questionableRegister = slot[Z].status.questionable.instrument.smu[X].enable
questionableRegister = slot[Z].status.questionable.instrument.smu[X].event
questionableRegister = slot[Z].status.questionable.instrument.smu[X].ntr
questionableRegister = slot[Z].status.questionable.instrument.smu[X].ptr
slot[Z].status.questionable.instrument.smu[X].enable = questionableRegister
slot[Z].status.questionable.instrument.smu[X].ntr = questionableRegister
slot[Z].status.questionable.instrument.smu[X].ptr = questionableRegister
```

questionableRegister	The status of the questionable status SMU summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number
X	Channel of the SMU: 1 or 2

Details

These attributes are used to read or write to the Questionable Status Instrument SMU Summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15. For example, if a value of 4352 is read as the value of the condition register, the binary equivalent is 0001 0001 0000 0000. This value indicates that bit B8 and bit B12 are set.

B15	B14	B13	B12	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0
**	>	>	>	>	>	>	>	>	>	>	>	>	>	>	*
0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0

* Least significant bit
** Most significant bit

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value and description
B0 to B7	Not used
B8	`slot[Z].status.questionable.instrument.smu[X].CAL` Set bit indicates that the calibration constants stored in nonvolatile memory were corrupted and could not be loaded when the instrument powered up. Bit B8 decimal value: 256
B9	Reserved for future use. Bit B9 decimal value: 512
B10	`slot[Z].status.questionable.instrument.smu[X].ACAL` An enabled bit in the questionable status automatic calibration expired summary event register is set. Bit B10 decimal value: 1,024
B11	Not used.
B12	`slot[Z].status.questionable.instrument.smu[X].OTEMP` Set bit indicates that an overtemperature condition was detected. Bit B12 decimal value: 4,096
B13 to B15	Not used

As an example, to set bit B8 of the questionable status SMU summary enable register for slot 1, set `slot[1].status.questionable.instrument.smu[1].enable = status.questionable.instrument.smu[1].CAL`.

In addition to the above constants, *`questionableRegister`* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *`questionableRegister`* to the sum of their decimal weights. For example, to set bits B8 and B12, set *`questionableRegister`* to 4352 (the sum of 256 + 4096).

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**

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	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^{9})	(2^{8})

Example

```
questionableRegister = slot[1].status.questionable.instrument.smu[1].CAL +
   slot[1].status.questionable.instrument.smu[1].OTEMP
slot[1].status.questionable.instrument.smu[1].enable = questionableRegister
```

Uses constants to set bit B8 and bit B12 of the questionable status SMU summary enable register for slot 1.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.operation.* (on page 244)

slot[Z].status.questionable.over_temperature.*

This attribute contains the Questionable Status Overtemperature Summary 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	6 (all bits set)

Usage

```
questionableRegister = slot[Z].status.questionable.over_temperature.condition
questionableRegister = slot[Z].status.questionable.over_temperature.enable
questionableRegister = slot[Z].status.questionable.over_temperature.event
questionableRegister = slot[Z].status.questionable.over_temperature.ntr
questionableRegister = slot[Z].status.questionable.over_temperature.ptr
slot[Z].status.questionable.over_temperature.enable = questionableRegister
slot[Z].status.questionable.over_temperature.ntr = questionableRegister
slot[Z].status.questionable.over_temperature.ptr = questionableRegister
```

questionableRegister	The status of the Questionable Status Overtemperature Summary register; a zero (0) indicates no bits set (also send 0 to clear all bits); other values indicate bits are set
Z	Module slot number

Details

These attributes are used to read or write to the questionable status over temperature summary registers. Reading a status register returns a value. The binary equivalent of the returned value indicates which register bits are set. The least significant bit of the binary number is bit B0, and the most significant bit is bit B15.

For information about .condition, .enable, .event, .ntr, and .ptr registers, refer to Status register set contents (on page 634). The individual bits of this register are defined in the following table.

Bit	Value
B0	Not used
B1	`slot[Z].status.questionable.over_temperature.SMU1` Set bit indicates that an overtemperature condition was detected on SMU 1. Bit B1 decimal value: 2 Binary value: 0000 0010
B2	`slot[Z].status.questionable.over_temperature.SMU2` Set bit indicates that an overtemperature condition was detected on SMU 2. Bit B2 decimal value: 4 Binary value: 0000 0100
B3 to B15	Not used

In addition to the above constants, *questionableRegister* can be set to the numeric equivalent of the bit to set. To set more than one bit of the register, set *operationRegister* to the sum of their decimal weights. For example, to set bits B1 and B2, set *questionableRegister* to 6 (the sum of 2 + 4).

Example

```
slot[2].status.questionable.over_temperature.enable = slot[2].status.questionable.
over_temperature.SMU1
```

Uses a constant to set the SMU 1 bit in the Questionable Status Overtemperature Summary enable register for slot 2.

Also see

Questionable Status Registers (on page 664), Questionable Status Registers (on page 673)
status.questionable.* (on page 434)

Trigger model TSP commands

Introduction

The TSP trigger model commands available for the MP5000 Series Modular Precision Mainframe and modules are listed in alphabetical order.

slot[Z].trigger.model.abort()

This function stops trigger model execution on the specified module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.abort("")
slot[Z].trigger.model.abort("triggerModelName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to abort

Details

This function stops the specified trigger model on the module. If a trigger model has been loaded, it will be unloaded during the abort sequence. If no trigger model name is provided, it aborts all trigger models running on the slot.

Example 1

```
-- Abort the trigger model "myTM" on slot 1.
slot[1].trigger.model.abort("myTM")
```

This example stops a trigger model that is running on the module in slot.

Example 2

```
-- Aborts all running trigger models on slot 1.
slot[1].trigger.model.abort()
```

This example shows how to abort all trigger models on the module in slot 1..

Also see

slot[Z].trigger.model.initiate() (on page 619)
slot[Z].trigger.model.load() (on page 620)
slot[Z].trigger.model.unload() (on page 624)

slot[Z].trigger.model.addblock.branch.always()

This function defines a trigger model block that always branches to a specific block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.addblock.branch.always("triggerModelName", "blockName", "branchToBlockName")`

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
branchToBlockName	Name of the block to branch to on true condition

Details

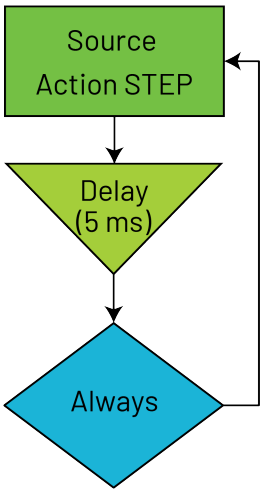
When a trigger model reaches a branch-always trigger block, it goes to the block specified by *branchToBlockName*. This block must exist in the trigger model.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(0, 5, 6)
-- Create a trigger model named myTM.
slot[1].trigger.model.delete("myTM")
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "SourceActionStep", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.005)
slot[1].trigger.model.addblock.branch.always("myTM", "", "SourceActionStep")
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Abort the trigger model after 1 second.
delay(1.0)
slot[1].trigger.model.abort("myTM")
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates a trigger model that uses `branch.always` to repeatedly step through a linear sweep on channel 1 until the trigger model is aborted.

Trigger model diagram



Also see

slot[Z].trigger.model.abort() (on page 570)
slot[Z].trigger.model.initiate() (on page 619)

slot[Z].trigger.model.addblock.branch.counter()

This function defines a trigger model block that branches to a block a specified number of times.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.branch.counter("triggerModelName", "blockName", "branchToBlockName",
    targetCount)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
branchToBlockName	Name of the block to branch to when the condition is true
targetCount	Number of times to repeat the branch

Details

This command defines a trigger model block that branches to another block a specified number of times.

The counter starts at 0 when the trigger model begins and increments each time this block is reached. The counter is then compared to the specified *targetCount*.

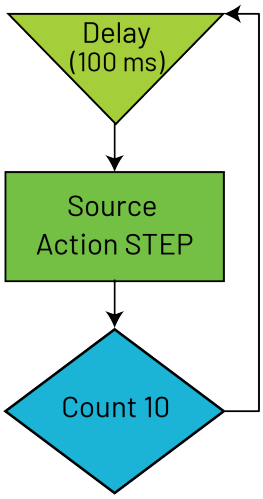
If the counter is less than or equal to *targetCount*, the trigger model branches to the specified *branchToBlockName* in the trigger model. If the counter exceeds *targetCount*, the trigger model continues to the next block in sequence, and the counter is reset to 0.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(0, 5, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.delay.constant("myTM", "DelayBlock", 0.1)
slot[1].trigger.model.addblock.source.action.step("myTM", "", 1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "DelayBlock", 10)
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates a trigger model that uses a branch counter block to step through a linear sweep on channel 1.

Trigger model diagram



Also see

None

slot[Z].trigger.model.addblock.branch.event()

This function defines a trigger model block that branches to another trigger model block when an event occurs.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.branch.event("triggerModelName", "blockName", "branchToBlockName",
    eventId)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
branchToBlockName	Name of the block to branch to when the condition is true
eventId	The event that must occur before the trigger model branches to the specified block; see **Details**

Details

This block goes to a specified trigger model block after a specified trigger event occurs. If the trigger event has not yet occurred when the trigger model reaches the `slot[Z].trigger.model.addblock.branch.event` block, the trigger model continues to execute the blocks in the normal sequence. After the trigger event occurs, the trigger model proceeds to the specified trigger model block when the `slot[Z].trigger.model.addblock.branch.event` block is encountered.

After an event occurs and this block executes, the branch is taken and the event detector is cleared. The trigger event must occur again for the trigger model to execute to the block again; otherwise, the blocks are executed in the normal sequence.

If you set the branch event to none, an error is generated when you run the trigger model.

Trigger models containing a branch on event block cannot also contain a measure overlapped block.

The following table shows the constants for the events.

ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8

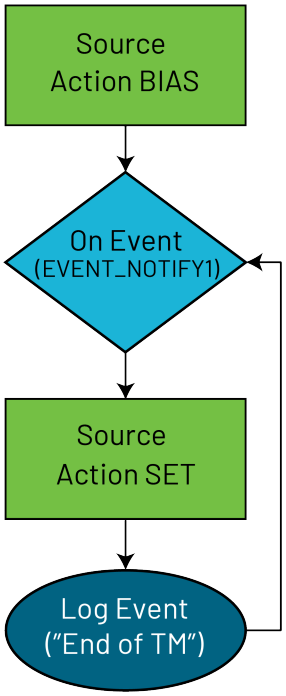
ID	Description
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(5, 5, 1)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.bias("myTM", "start", 1)
slot[1].trigger.model.addblock.branch.event("myTM", "", "end", slot[1].trigger.mod
el.EVENT_NOTIFY1)
slot[1].trigger.model.addblock.source.action.set("myTM", "start", 1)
slot[1].trigger.model.addblock.logevent("myTM", "end", slot[1].trigger.model.LOG_I
NFO1, "End of Trigger Model")
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This trigger model starts at bias level and sets 5 V unless EVENT_NOTIFY1 occurs, in which case it skips the voltage change and logs an event.

Trigger model diagram



Also see

Branch on event block (on page 76)

slot[Z].trigger.model.addblock.branch.once()

This function defines a trigger model block that branches to a specified trigger model block the first time it is encountered in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.branch.once("triggerModelName", "blockName", "branchToBlockName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block; an empty string is a valid block name
branchToBlockName	Name of the block to branch to on true condition

Details

This block causes the trigger model to branch to the specified *branchToBlockName* the first time it is encountered during execution. On subsequent passes, that block is skipped and the trigger model proceeds to the next block.

To reset this block to execute again, you must stop execution and initiate the trigger model again.

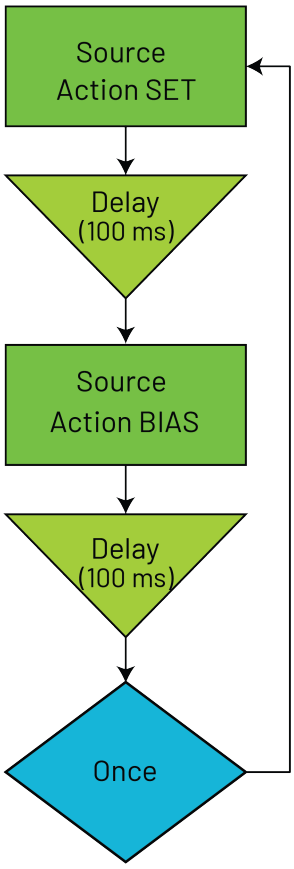
The block specified by *branchToBlockName* must exist in the trigger model before running.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(5, 5, 1)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.set("myTM", "start", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.once("myTM", "", "start")
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
--Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates how to create two pulses using branch.once in a trigger model.

Trigger model diagram



Also see

Branch once block (on page 76)
slot[Z].trigger.model.addblock.branch.onceexcluded() (on page 577)

slot[Z].trigger.model.addblock.branch.onceexcluded()

This function causes the trigger model to branch to a specified trigger model block every time it is encountered in the trigger model except for the first time.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.branch.onceexcluded("triggerModelName", "blockName",
    "branchToBlockName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block; an empty string is a valid block name
branchToBlockName	Name of the block to branch to on true condition

Details

This block is skipped the first time the trigger model encounters it; the trigger model proceeds to the next block. After the first encounter, the trigger model always branches to the specified block.

This block resets automatically each time the trigger model is initiated.

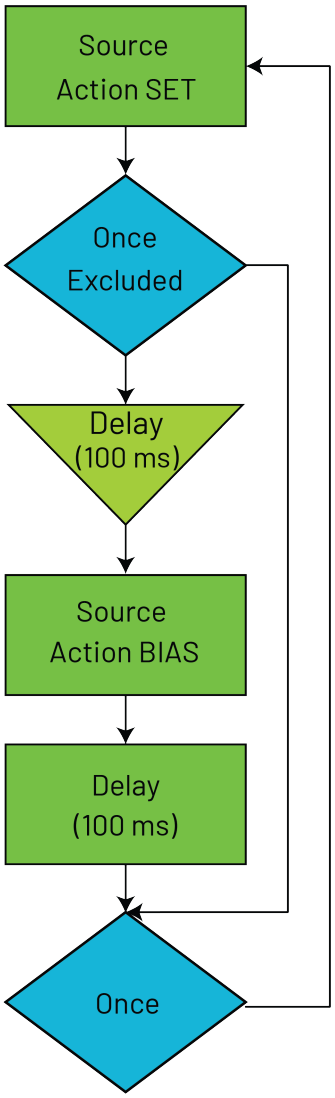
The block specified by *branchToBlockName* must exist in the trigger model before running.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(5, 5, 1)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.set("myTM", "start", 1)
slot[1].trigger.model.addblock.branch.onceexcluded("myTM", "", "end")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.once("myTM", "end", "start")
-- Turn on the output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
--Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates how to create a trigger model that pulses once, then stays at 5 V and exits using branch.onceexcluded.

Trigger model diagram



Also see

Branch once excluded block (on page 76)
slot[Z].trigger.model.addblock.branch.once() (on page 576)

slot[Z].trigger.model.addblock.configlist.next()

This function defines a trigger model block that recalls the settings stored at the next index of a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.configlist.next("triggerModelName", "blockName", channel,
    "configlistName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
channel	Module channel number; see **Details**.
configlistName	The name of the configuration list to recall

Details

When the trigger model reaches this block, the settings at the next index in the specified configuration list are restored.

On its first use in a trigger model, this block recalls index 1 unless an earlier recall block has already recalled a specific index. In that case, the block continues from that point and recalls the next index instead.

When the end of the list is reached, the trigger model starts again from index 1.

The channels are specified as either integers or slot notation *Z*00*X*, where *Z* is the slot number and *X* is the channel number. For example, channel 1 on slot 1 is specified as either 1 or 1001.

The configuration list must be defined before the trigger model uses this block.

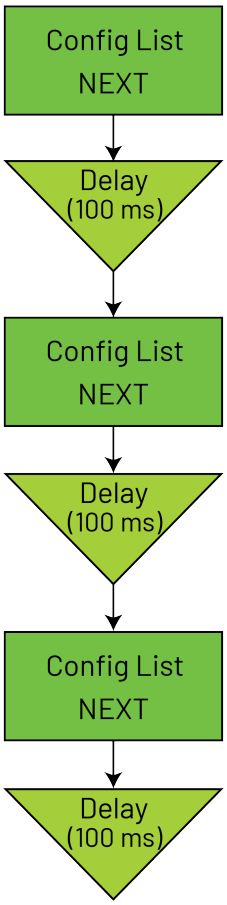
Example

```
-- Create and store a 3-point configuration list.
slot[1].psu[1].configlist.create("myConfigList")
slot[1].psu[1].source.levelv = 1.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 2.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 3.0
slot[1].psu[1].configlist.store("myConfigList")
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.configlist.next("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.configlist.next("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.configlist.next("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates how to use the `configlist.next()` trigger model block to cycle through a 3-point configuration list.

The channel output steps through three voltage levels (1.0 V, 2.0 V, and 3.0 V) using three consecutive `configlist.next()` blocks with delays between each step.

Trigger model example



Also see

Configuration list next block (on page 71)
slot[Z].trigger.model.addblock.configlist.prev() (on page 582)
slot[Z].trigger.model.addblock.configlist.recall() (on page 585)

slot[Z].trigger.model.addblock.configlist.prev()

This function defines a trigger model block that recalls the settings that are stored in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.configlist.prev("triggerModelName", "blockName", channel,
    "configlistName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block will be added
blockName	Name of the block to be added; an empty string is a valid block name
channel	Module channel number; see **Details**
configlistName	A string that defines the configuration list to recall

Details

When the trigger model reaches a configuration list previous block, the settings for the previous index in the specified configuration list are restored.

On the first use in a trigger model, this block recalls the last index unless an earlier recall block has already recalled a specific index. In that case, the block will continue from that point and recall the previous index instead.

When the beginning of the list is reached, the trigger model continues from the last index in the list.

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

The configuration list must be defined before the trigger model uses this block.

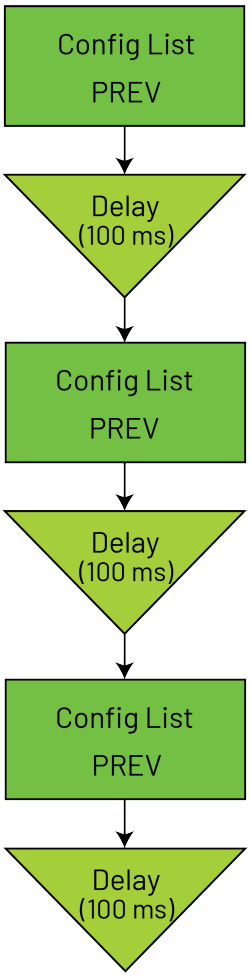
Example

```
-- Create and store a 3-point configuration list.
slot[1].psu[1].configlist.create("myConfigList")
slot[1].psu[1].source.levelv = 1.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 2.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 3.0
slot[1].psu[1].configlist.store("myConfigList")
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.configlist.prev("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.configlist.prev("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.configlist.prev("myTM", "", 1, "myConfigList")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
--Turn the source off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates how to use the `configlist.prev()` trigger model block to cycle through a 3-point configuration list.

The channel output steps through three voltage levels (3.0 V, 2.0 V, and 1.0 V) using three consecutive `configlist.prev()` blocks with delays between each step.

Trigger model diagram



Also see

Configuration list recall previous block (on page 71)
slot[Z].trigger.model.addblock.configlist.next() (on page 580)
slot[Z].trigger.model.addblock.configlist.recall() (on page 585)

slot[Z].trigger.model.addblock.configlist.recall()

This function recalls the settings that are stored in a configuration list.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.configlist.recall("triggerModelName", "blockName", channel,
    "configlistName")
slot[Z].trigger.model.addblock.configlist.recall("triggerModelName", "blockName", channel,
    "configlistName", index)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
channel	Module channel number; see **Details**
configlistName	A string that defines the configuration list to recall
index	The index in the configuration list to recall; defaults to 1 if no index is specified

Details

When the trigger model reaches a configuration list recall block, the settings in the specified configuration list are recalled.

You can recall a specific set of configuration settings in the configuration list by specifying its index.

The configuration list must be defined before you can use this block. If one of the indexes for the configuration list changes (because of an item being added or removed from a configuration list), verify that the trigger model count is still accurate.

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

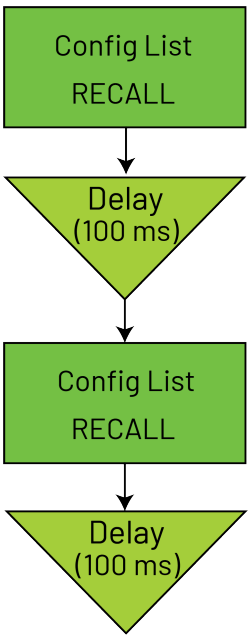
Example

```
-- Create and store a configuration list with 3 indexes.
slot[1].psu[1].configlist.create("myConfigList")
slot[1].psu[1].source.levelv = 1.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 2.0
slot[1].psu[1].configlist.store("myConfigList")
slot[1].psu[1].source.levelv = 3.0
slot[1].psu[1].configlist.store("myConfigList")
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.configlist.recall("myTM", "", 1, "myConfigList", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.configlist.recall("myTM", "", 1, "myConfigList", 3)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
--Turn output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates how to use the `configlist.recall()` trigger model block to cycle through a configuration list with 3 indexes.

The channel output steps through two voltage levels (1.0 V and 3.0 V) using two `configlist.recall()` blocks with delays between each step.

Trigger model diagram



Also see

Configuration list recall block (on page 71)
slot[Z].trigger.model.addblock.configlist.next() (on page 580)
slot[Z].trigger.model.addblock.configlist.prev() (on page 582)

Configuration list commands

slot[Z].psu[X].configlist.create() (on page 372)
slot[Z].smu[X].configlist.create() (on page 476)
slot[Z].psu[X].configlist.delete() (on page 373)
slot[Z].smu[X].configlist.delete() (on page 478)
slot[Z].psu[X].configlist.recall() (on page 376)
slot[Z].smu[X].configlist.recall() (on page 480)
slot[Z].psu[X].configlist.size() (on page 377)
slot[Z].smu[X].configlist.size() (on page 481)
slot[Z].psu[X].configlist.store() (on page 378)
slot[Z].smu[X].configlist.store() (on page 482)
slot[Z].psu[X].configlist.table() (on page 379)
slot[Z].smu[X].configlist.table() (on page 483)

slot[Z].trigger.model.addblock.delay.constant()

This function defines a trigger model block that adds a constant delay to the execution of a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.delay.constant("triggerModelName", "blockName", delayTime)
slot[Z].trigger.model.addblock.delay.constant("triggerModelName", "blockName", delayTime,
    "referenceBlockName")
slot[Z].trigger.model.addblock.delay.constant("triggerModelName", "blockName", {delayList})
slot[Z].trigger.model.addblock.delay.constant("triggerModelName", "blockName", {delayList},
    "referenceBlockName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added
delayTime	Time delay as a positive value: 0.000001 s to 1,000,000 s
referenceBlockName	Trigger model block to use for the start time of the delay; see **Details**
delayList	Table of delay intervals in seconds; see **Details**

Details

When the trigger model reaches this block, it pauses for a specified time. If *referenceBlockName* is specified, the trigger model pauses for the time delay since the start of the reference block.

If a reference block was not visited before the delay block, the entry time of the trigger model is used.

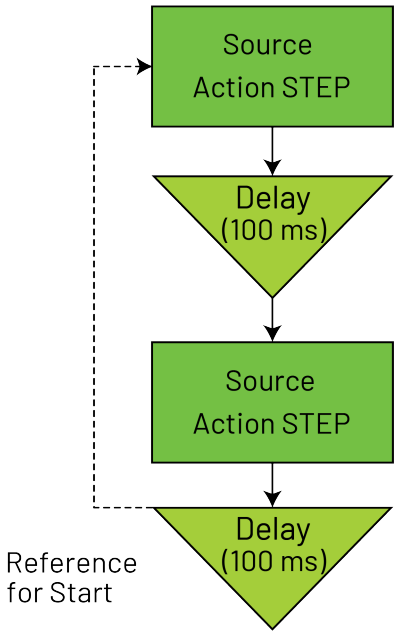
When *delayList* is defined, for each time the delay block is encountered in the trigger model, the next value in the list is used as the delay time. After all elements in the array have been used, the delays restart at the beginning of the list.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(2, 4, 2)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.step("myTM", "Step2", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1, "Step2")
-- Turn on output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates a trigger model that performs a source action step, delays 100 ms, performs a second step, and then delays another 100 ms (second delay is referenced to the start of the second step).

Trigger model diagram



Also see

None

slot[Z].trigger.model.addblock.logevent()

This function defines a trigger model block that logs an event in the event log.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.addblock.logevent("triggerModelName", "blockName", eventType, "message")`

Z	Module slot number
triggerModelName	Name of the trigger model to which the block will be added
blockName	Name of the block to be added; an empty string is a valid block name
eventType	The type of event and event number; see **Details**
message	Message to display in the event log, up to 52 characters

Details

This block allows you to log an event in the event log when trigger model execution reaches this block. You can also force the trigger model to abort with this block. When the trigger model executes the block, the defined event is logged. If the abort option is selected, the trigger model is also aborted immediately.

The *eventType* parameter defines the type of event:

- Information: `slot[Z].trigger.model.LOG_INFO`*N*
- Warning: `slot[Z].trigger.model.LOG_WARN`*N*
- Error: `slot[Z].trigger.model.LOG_ERROR`*N*
- Abort: `slot[Z].trigger.model.LOG_WARN_ABORT`

Where *N* is 1 to 4. You can define up to four events of each type.

You can also set *eventNumber* to `trigger.model.LOG_WARN_ABORT`, which aborts the trigger model immediately and posts a warning event log message.

All events generated by this block are logged in the event log. Warning and error events are also displayed briefly on the front-panel display.

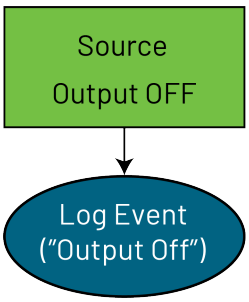
If you use this block too often in a trigger model, it could overflow the event log. The log event block takes additional time to record the event and may impact timing in critical paths. Avoid using this block in time-sensitive sections of the trigger model, such as when there must be a precise delay between two trigger model blocks or actions.

Example

```
-- Create the trigger model
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].OFF)
slot[1].trigger.model.addblock.logevent("myTM", "", slot[1].trigger.model.LOG_INFO
1, "Output Off")
-- Initiate the trigger model
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete
waitcomplete()
```

This example logs an information message (`Output Off`) immediately after the output is turned off by the source output block.

Trigger model diagram



Also see

Log event block (on page 74)

slot[Z].trigger.model.addblock.measure()

This function defines a trigger block that makes a measurement.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.measure("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.measure("triggerModelName", "blockName", channel, count)
slot[Z].trigger.model.addblock.measure("triggerModelName", "blockName", {channelList})
slot[Z].trigger.model.addblock.measure("triggerModelName", "blockName", {channelList}, count)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block; an empty string is a valid block name
channel	Channel to make the measurement on; see **Details**
count	Number of measurements to make when this block is executed; if not defined, the configured measurement count of the channel is used
channelList	Table of channels to make measurements on; for example: `{1, 2}`; see **Details**

Details

This block triggers measurements based on the measurement function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

- The instrument begins making measurements
- The trigger model waits for the measurement to be made

You can specify one or more channels for the measurement to run on using *channelList*. All measurements run all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

All specified channels must be present on the module running the trigger model. If more than one channel is specified, the measure count for each channel must be the same unless a count value is also specified.

If *`count`* is specified, it overrides the measure count configured for the channels by `slot[`*`Z`*`].psu[`*`X`*`].measure.count()` or `slot[`*`Z`*`].smu[`*`X`*`].measure.count()`.

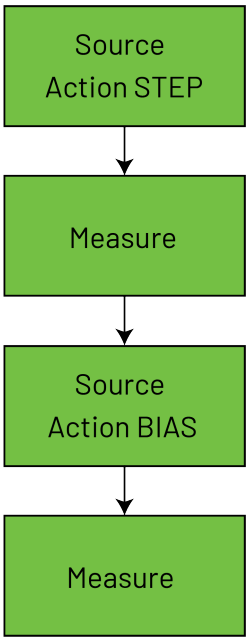
If you are using a SMU in your trigger model, configured measure delays are imposed when the SMU encounters the measure action block.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(5, 5, 1)
-- Configure the measurement.
READING_BUFFER = slot[1].psu[1].makebuffer(10)
slot[1].psu[1].trigger.measure.v(READING_BUFFER)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "", 1)
slot[1].trigger.model.addblock.measure("myTM", "", 1, 1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.measure("myTM", "", 1, 1)
-- Turn on the output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
-- Print the measurement results.
printbuffer(1, READING_BUFFER.n, READING_BUFFER)
```

This example demonstrates a trigger model that captures one measurement for source step and another measurement for source bias in a linear sweep.

Trigger model diagram



Also see

Measure block (on page 71)
slot[Z].psu[X].measure.count() (on page 382)
slot[Z].smu[X].measure.count() (on page 496)

slot[Z].trigger.model.addblock.measureoverlapped()

This function defines a trigger model block that starts a measurement in an overlap state (in the background), allowing the trigger model to continue execution without waiting for the measurement to complete.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.measureoverlapped("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.measureoverlapped("triggerModelName", "blockName", channel, count)
slot[Z].trigger.model.addblock.measureoverlapped("triggerModelName", "blockName", {channelList})
slot[Z].trigger.model.addblock.measureoverlapped("triggerModelName", "blockName", {channelList},
    count)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block; an empty string is a valid block name
channel	Channel to make the measurement on; see **Details**
count	Number of measurements to make when this block starts
channelList	Table of channels to make measurements on; for example: `{channel1, channel2,...}`; see **Details**

Details

This command is used only with SMU instruments.

This block triggers measurements based on the measure function that is selected when the trigger model is initiated. When trigger model execution reaches this block:

- The instrument begins making measurements
- The instrument processes the reading and places it into the specified reading buffer. In the trigger model, the reading buffer must be specified in the slot[Z].smu[X].trigger.measureY() (on page 530) or slot[Z].psu[X].trigger.measureY() (on page 403) measurement configuration commands. New reading buffers must be created before these commands are called.

You can specify one or more channels for the measurement to run on using *channelList*. All measurements run on all specified channels at the same time, plus or minus 500 ns.

The channels are specified as either integers or slot notation ZOOX, where Z is the slot number and X is the channel number. For example, channel 1 on slot 1 is specified as either 1 or 1001.

All channels must be present on the same module running the trigger model. Only one measure overlapped block may exist on a channel; there may be no other measure or measure overlapped blocks for that channel.

The channels are specified as either integers or slot notation ZOOX, where Z is the slot number and X is the channel number. For example, channel 1 on slot 1 is specified as either 1 or 1001.

When running the trigger model with the measure overlapped block, overlapped measurements can only be made with source autoranging turned off. To turn the source autoranging off, set the slot[*Z*].smu[*X*].source.autoranging*Y* attribute to slot[*Z*].smu[*X*].OFF.

The source limit must be fixed during the running of the trigger model, and it cannot be altered by the configuration list.

Trigger models containing a measure overlapped block cannot also contain a branch on event block.

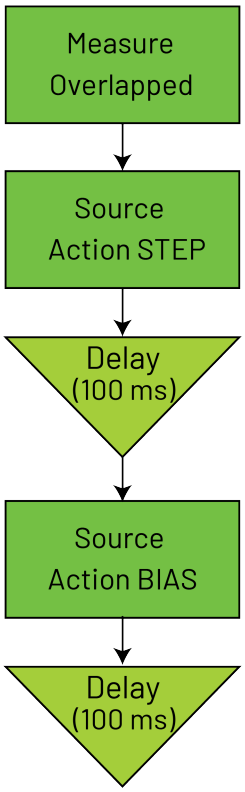
If any of the above items are incorrectly configured, then the slot[z].smu[x].trigger.model.initiate() function generates an error.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.linearv(5, 5, 1)
-- Configure the source range.
slot[1].smu[1].source.rangev = 6
-- Configure the measurement.
READING_BUFFER = slot[1].smu[1].makebuffer(10)
slot[1].smu[1].trigger.measure.v(READING_BUFFER)
slot[1].smu[1].measure.aperture = 0.02
-- Create the trigger model
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.measureoverlapped("myTM", "", 1, 10)
slot[1].trigger.model.addblock.source.action.step("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
-- Turn on output before initiating the trigger model
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete
waitcomplete()
slot[1].smu[1].source.output = slot[1].smu[1].OFF
-- Print the measurement results
printbuffer(1, READING_BUFFER.n, READING_BUFFER)
```

This example demonstrates how to use the measureoverlapped block in a trigger model to perform 10 overlapped voltage measurements on SMU channel 1. Source range must be fixed when performing measure overlapped.

Trigger model diagram



Also see

Measure overlapped block (on page 72)
slot[Z].smu[X].source.autorangeY (on page 511)
slot[Z].smu[X].source.limitnY (on page 515)
slot[Z].smu[X].source.limitpY (on page 516)
slot[Z].smu[X].source.limitY (on page 517)

slot[Z].trigger.model.addblock.nop()

This function creates a placeholder block that performs no action in the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.nop("triggerModelName", "blockName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block will be added
blockName	Name of the trigger block; an empty string is a valid block name

Details

This trigger block creates a placeholder that performs no action in the trigger model.

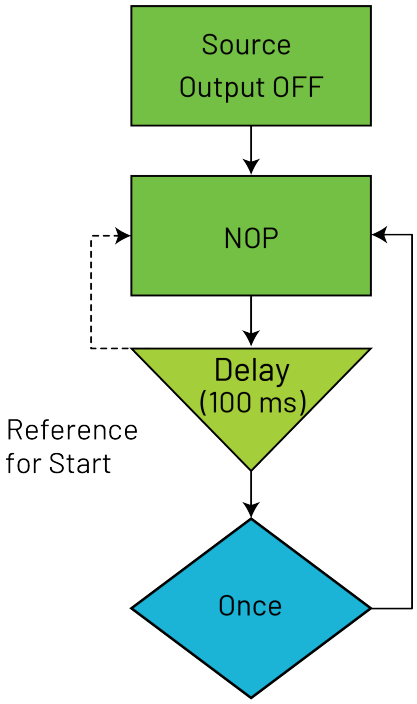
You can use this command to create a named block from which a trigger model branches forward or backward.

Example

```
-- Configure the source level.
slot[1].smu[1].source.levelv = 3
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].smu[1].OFF)
slot[1].trigger.model.addblock.nop("myTM", "nop2")
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1, "nop2")
slot[1].trigger.model.addblock.branch.once("myTM", "", "nop2")
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
```

This example demonstrates a trigger model that uses a no operation block for a delay timing reference and for a branch target by branch.counter.

Trigger model diagram



Also see

No output block (on page 72)

slot[Z].trigger.model.addblock.notify()

This function defines a trigger model block that generates a trigger event.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.addblock.notify("triggerModelName", "blockName", eventId)`

Z	Module slot number
triggerModelName	Name of the trigger model to which the block will be added
blockName	Name of the trigger block; an empty string is a valid block name
eventId	`slot[Z].trigger.model.EVENT_NOTIFYN`, where *N* is the number of the notify event; see **Details**

Details

When trigger model execution reaches this block, the instrument generates a trigger event and immediately continues to the next block.

Other commands can reference the event that the notify block generates. For example, you can use the notify event as the stimulus of a hardware trigger line or the gating event for another trigger wait block to continue execution after waiting on the specific event generated by this notification.

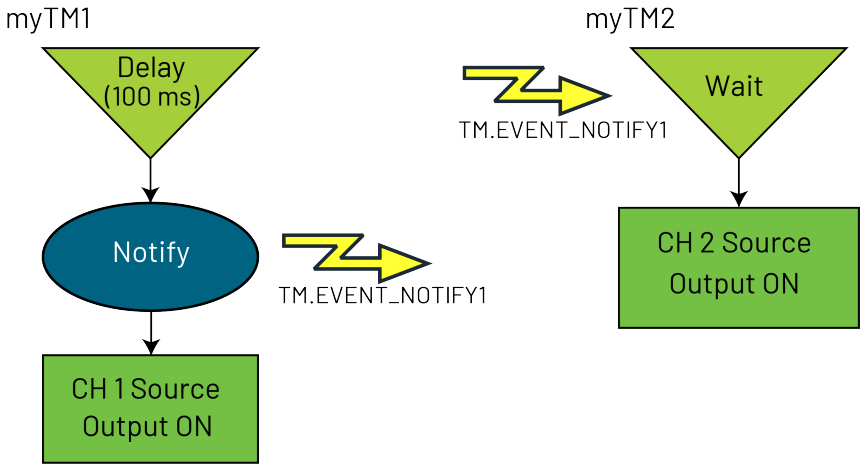
You can define 1 to 16 notify events for PSU modules. You can define 1 to 8 for SMU modules.

Example

```
-- Create the trigger model myTM1.
slot[1].trigger.model.create("myTM1")
slot[1].trigger.model.addblock.delay.constant("myTM1", "", 0.1)
slot[1].trigger.model.addblock.notify("myTM1", "", slot[1].trigger.model.EVENT_NOT
IFY1)
slot[1].trigger.model.addblock.source.output("myTM1", "", 1, slot[1].psu[1].ON)
-- Create the trigger model myTM2.
slot[1].trigger.model.create("myTM2")
slot[1].trigger.model.addblock.wait("myTM2", "", slot[1].trigger.model.EVENT_NOTIF
Y1)
slot[1].trigger.model.addblock.source.output("myTM2", "", 2, slot[1].psu[2].ON)
-- Configure the source level.
slot[1].psu[1].source.levelv = 5
slot[1].psu[2].source.levelv = 5
-- Initiate the trigger models.
slot[1].trigger.model.initiate("myTM2")
slot[1].trigger.model.initiate("myTM1")
-- Wait for the trigger model to complete.
waitcomplete()
--Turn the outputs off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
slot[1].psu[2].source.output = slot[1].psu[2].OFF
```

This example demonstrates how to use the notify block in a trigger model to create a simple delay before turning on the output of channel 1. Trigger model `myTM1` waits for 0.1 s before it notifies `myTM2` to turn on the output of channel 2.

Trigger model diagram



Also see

Notify blocks (on page 74)
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)

slot[Z].trigger.model.addblock.reset.branch.counter()

This function defines a trigger model block that resets the count for a branch counter block.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.reset.branch.counter("triggerModelName", "blockName",
    "resetBranchCountBlockName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
resetBranchCountBlockName	Name of the branch counter block to reset the count value to 0

Details

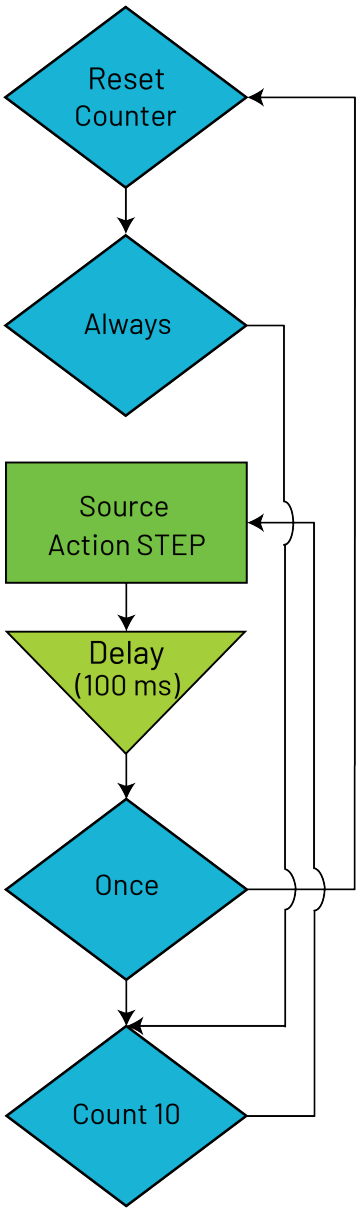
This command defines a trigger model building block that resets the count of the branch counter block specified by *resetBranchCountBlockName*.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(0, 5, 11)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.reset.branch.counter("myTM", "resetCount",
"branchCounter")
slot[1].trigger.model.addblock.branch.always("myTM", "", "branchCounter")
slot[1].trigger.model.addblock.source.action.step("myTM", "step", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.once("myTM", "", "resetCount")
slot[1].trigger.model.addblock.branch.counter("myTM", "branchCounter", "step", 10)
-- Turn on the output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates a trigger model using the reset.branch.counter block, where the branch counter resets to zero each time it branches, resulting in the model stepping 10 times.

Trigger model diagram



Also see

Reset branch count block (on page 76)
slot[Z].trigger.model.addblock.branch.counter() (on page 572)

slot[Z].trigger.model.addblock.source.action.bias()

This function defines a trigger model block that sets the source output level to the bias level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.bias("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.source.action.bias("triggerModelName", "blockName", {channelList})
```

Z	Module slot number
triggerModelName	Name of the trigger model to which this block is added
blockName	Name of the trigger block; an empty string is a valid block name
channel	Channel number; see **Details**
channelList	Table of channels to make measurements on; for example: `{1, 2}`; see **Details**

Details

The bias level is defined by the `slot[Z].psu[X].source.levelv` or `slot[Z].smu[X].source.levelv` commands, which must be configured before the trigger model starts. When this block is reached, the source output is set to the bias level. For example, if you want to return to 2 V on each step, set the source level to 2 V, then define the action bias block as part of the trigger model.

Use this command with the `slot[Z].trigger.model.addblock.source.action.step()` command to create pulses that go to the voltage in the sweep array defined by any of the following commands and then to the bias voltage:

- `slot[Z].psu[X]trigger.source.linearv()`
- `slot[Z].smu[X]trigger.source.linearv()`
- `slot[Z].psu[X].trigger.source.listv`
- `slot[Z].smu[X].trigger.source.listv`
- `slot[Z].psu[X].trigger.source.logv`
- `slot[Z].smu[X].trigger.source.logv`

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

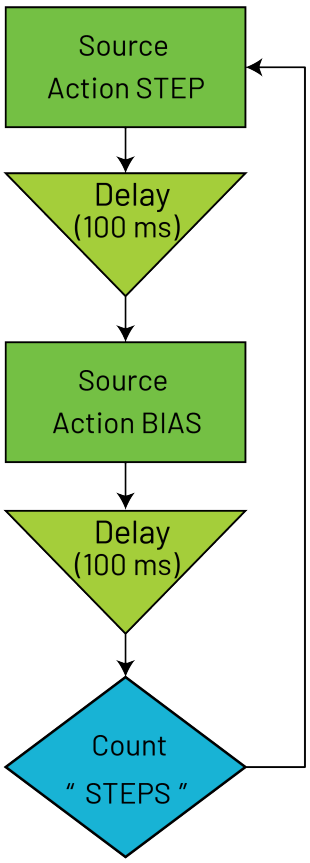
All specified channels must be on the module that is running the trigger model.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.linearv(0, 5, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "step", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "step", 10)
-- Turn on the output before the trigger model is initiated.
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model that performs a linear voltage sweep with 10 steps. After each step, it delays 100 ms, returns to the bias level, delays another 100 ms, and repeats this sequence for all steps, resulting in 10 pulses with 100 ms intervals.

Trigger model diagram



Also see

slot[Z].psu[X].source.levelv (on page 392)
slot[Z].smu[X].source.levelY (on page 514)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

Source action bias block (on page 72)

Sweep configuration commands

slot[Z].psu[X].trigger.source.linearY() (on page 405)
slot[Z].smu[X].trigger.source.linearY() (on page 532)
slot[Z].psu[X].trigger.source.listY() (on page 406)
slot[Z].smu[X].trigger.source.listY() (on page 533)
slot[Z].psu[X].trigger.source.logY() (on page 408)
slot[Z].smu[X].trigger.source.logY() (on page 535)

slot[Z].trigger.model.addblock.source.action.pulseset()

This function defines a trigger model block that sets the source output pulse to the value at the present index in the list of sweep values without advancing the index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.pulseset("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.source.action.pulseset("triggerModelName", "blockName",
    {channelList})
```

Z	Module slot number
triggerModelName	Name of the trigger model to which this block is added
blockName	Name of the trigger block; an empty string is a valid block name
channel	Channel number; see **Details**
channelList	Table of channels to make measurements on; for example: {1, 2}; see **Details**

Details

This block sets the source output pulse to the sweep value at the present index of the sweep table without advancing to the next point. This allows you to apply the same value across multiple blocks or delay periods without changing the sweep position.

This block can be used in a trigger model to reapply or hold a source value for a specified time before proceeding. This command does not increment the sweep index, so repeated executions use the same value unless the index is modified using another action block.

The list of sweep values is created by one of the following commands:

- slot[Z].smu[X].trigger.source.linearY
- slot[Z].smu[X].trigger.source.listY
- slot[Z].smu[X].trigger.source.logY

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing. The channels are specified as either integers or slot notation Z00X, where Z is the slot number and X is the channel number. For example, channel 1 on slot 1 is specified as either 1 or 1001.

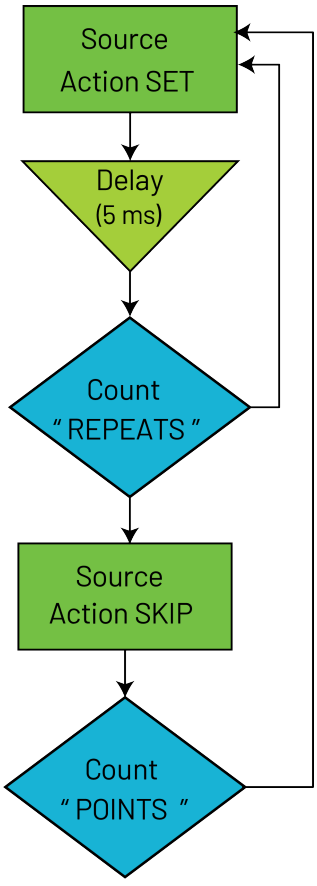
All specified channels must be on the module running the trigger model.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.lineari(0, 1, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.pulseset("myTM", "PulseSet", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.005)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "PulseSet", 3)
slot[1].trigger.model.addblock.source.action.skip("myTM", "", 1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "PulseSet", 5)
-- Turn on the output before the trigger model is initiated.
slot[1].smu[1].source.leveli = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example applies each step of a linear current sweep three times before advancing. The `slot[Z].trigger.model.addblock.source.action.pulseset()` command holds the current sweep value and repeats it with `slot[Z].trigger.model.addblock.branch.counter()` before using `slot[Z].trigger.model.addblock.source.action.skip()` to move to the next value.

Trigger model diagram



Also see

slot[Z].smu[X].source.pulse.limitnY (on page 525)
slot[Z].smu[X].source.pulse.limitpY (on page 526)
slot[Z].smu[X].source.pulse.limitY (on page 527)
slot[Z].trigger.model.addblock.source.action.pulsestep() (on page 604)

Sweep configuration commands

slot[Z].smu[X].trigger.source.linearY (on page 532)
slot[Z].smu[X].trigger.source.listY (on page 533)
slot[Z].smu[X].trigger.source.logY (on page 535)

slot[Z].trigger.model.addblock.source.action.pulsestep()

This function defines a trigger model block that advances the index in the list of sweep values for the specified channel and then sets the source pulse value to the value at that index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.pulsestep("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.source.action.pulsestep("triggerModelName", "blockName",
    {channelList})
```

Z	Module slot number
triggerModelName	Name of the trigger model to which this block is added
blockName	Name of the trigger block; an empty string is a valid block name
channel	Channel number; see **Details**
channelList	Table of channels to make measurements on; for example: {1, 2}; see **Details**

Details

This command advances the source output pulse for the specified channel to the next value in the pre-defined sweep list. The sweep list must be created before running the trigger model using one of the supported sweep configuration commands.

- slot[Z].smu[X].source.pulse.limitnY
- slot[Z].smu[X].source.pulse.limitpY
- slot[Z].smu[X].source.pulse.limitY

Each time the block is executed, the sweep index increases by one, updating the output to the corresponding sweep value. Once the end of the sweep list is reached, it starts again at the beginning.

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

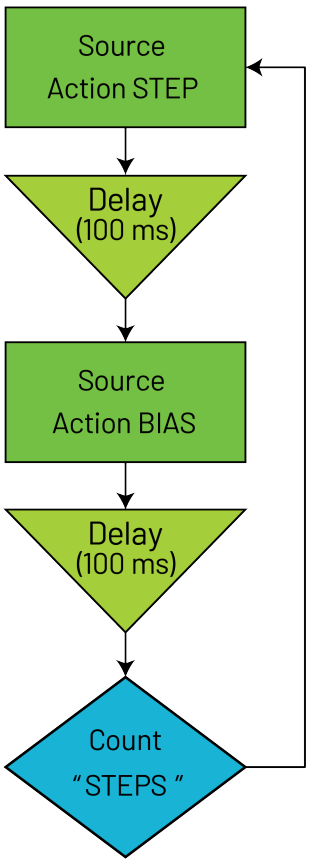
The channels are specified as either integers or slot notation ZOOX, where Z is the slot number and X is the channel number. For example, channel 1 on slot 1 is specified as either 1 or 1001. All specified channels must be on the module running the trigger model.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.lineari(0, 1, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.pulsestep("myTM", "pulsestep", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "pulsestep", 10)
-- Turn on the output before the trigger model is initiated.
slot[1].smu[1].source.leveli = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model that performs a linear current sweep with 10 steps. After each step, there is a 100 ms delay, then a return to the bias level, another 100 ms delay, and so on until the steps are completed.

Trigger model diagram



Also see

slot[Z].smu[X].source.pulse.limitnY (on page 525)
slot[Z].smu[X].source.pulse.limitpY (on page 526)
slot[Z].smu[X].source.pulse.limitY (on page 527)

slot[Z].trigger.model.addblock.source.action.pulseset() (on page 602)

Sweep configuration commands

slot[Z].smu[X].trigger.source.linearY (on page 532)
slot[Z].smu[X].trigger.source.listY (on page 533)
slot[Z].smu[X].trigger.source.logY (on page 535)

slot[Z].trigger.model.addblock.source.action.set()

This function defines a trigger model block that sets the source output level to the value at the present index in the list of sweep values without advancing the index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.set("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.source.action.set("triggerModelName", "blockName", {channelList})
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the block to be added; an empty string is a valid block name
channel	Channel number; see **Details**
channelList	Table of channels to make measurements on; for example: `{channel1, channel2, ...}`. See **Details**.

Details

This block sets the source output to the sweep value at the present index of the sweep table without advancing to the next point. This allows you to apply the same value across multiple blocks or delay periods without changing the sweep position.

This block can be used in a trigger model to reapply or hold a source value for a specified time before proceeding. This command does not increment the sweep index, so repeated executions use the same value unless the index is modified using another action block.

The list of sweep values is created by one of the following commands:

- `slot[Z].smu[X].trigger.source.linearY`
- `slot[Z].smu[X].trigger.source.listY`
- `slot[Z].smu[X].trigger.source.logY`
- `slot[Z].psu[X].trigger.source.linearY`
- `slot[Z].psu[X].trigger.source.listY`
- `slot[Z].psu[X].trigger.source.logY`

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

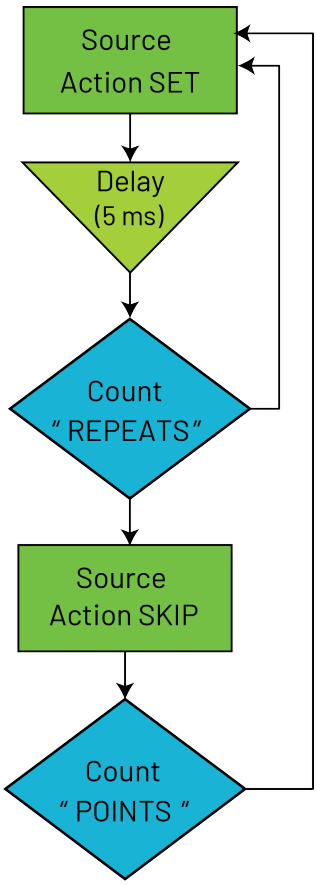
All specified channels must be on the module running the trigger model.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(0, 5, 5)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.set("myTM", "set", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.005)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "set", 3)
slot[1].trigger.model.addblock.source.action.skip("myTM", "", 1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "set", 5)
-- Turn on the output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This trigger model applies each step of a linear voltage sweep three times before advancing. It uses source.action.set to hold the current sweep value and repeats it with branch.counter before using source.action.skip to move to the next value.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)

Source action set block (on page 73)

Sweep configuration commands

slot[Z].psu[X].trigger.source.linearY (on page 405)
slot[Z].psu[X].trigger.source.listY() (on page 406)
slot[Z].psu[X].trigger.source.logY() (on page 408)
slot[Z].smu[X].trigger.source.linearY (on page 532)
slot[Z].smu[X].trigger.source.listY() (on page 533)
slot[Z].smu[X].trigger.source.logY() (on page 535)

slot[Z].trigger.model.addblock.source.action.skip()

This function defines a trigger model block that advances the index in the list of sweep values with no change to the source level.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.skip("triggerModelName", "blockName", channel)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block to branch to; an empty string is a valid block name
channel	Channel number; see **Details**

Details

This command advances the index in the pre-defined list of sweep values with no change to the source level. The sweep list must be created before running the trigger model using one of the supported sweep configuration commands:

- `slot[Z].smu[X].trigger.source.linearY`
- `slot[Z].smu[X].trigger.source.listY`
- `slot[Z].smu[X].trigger.source.logY`
- `slot[Z].psu[X].trigger.source.linearY`
- `slot[Z].psu[X].trigger.source.listY`
- `slot[Z].psu[X].trigger.source.logY`

Each time this block is executed, the index is advanced one spot in the sweep list. Once the end of the sweep list is reached, the block returns to the beginning of the list and starts again.

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

The channels are specified as either integers or slot notation `Z00X`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

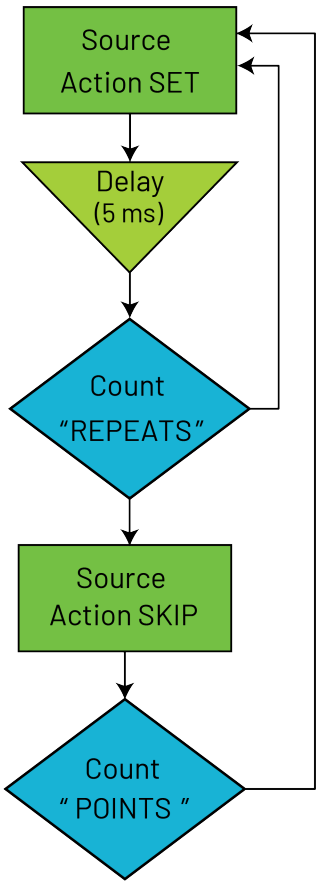
All specified channels must be present on the module running the trigger model.

Example

```
-- Configure the sweep.
slot[1].psu[1].trigger.source.linearv(0, 5, 5)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.set("myTM", "set", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.005)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "set", 3)
slot[1].trigger.model.addblock.source.action.skip("myTM", "", 1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "set", 5)
-- Turn on the output before the trigger model is initiated.
slot[1].psu[1].source.levelv = 0
slot[1].psu[1].source.output = slot[1].psu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example demonstrates a trigger model that performs a linear sweep over five points. For each sweep point, it repeats the same step three times before advancing to the next point. This sequence is repeated for all sweep points.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)
slot[Z].trigger.model.addblock.source.action.step() (on page 610)
slot[Z].trigger.model.addblock.source.output() (on page 612)

Source action skip block (on page 73)

Sweep configuration commands

slot[Z].psu[X].trigger.source.linearY (on page 405)
slot[Z].smu[X].trigger.source.linearY (on page 532)
slot[Z].psu[X].trigger.source.listY (on page 406)
slot[Z].smu[X].trigger.source.listY (on page 533)
slot[Z].psu[X].trigger.source.logY (on page 408)
slot[Z].smu[X].trigger.source.logY (on page 535)

slot[Z].trigger.model.addblock.source.action.step()

This function defines a trigger model block that advances the index in the list of sweep values for the specified channel and then sets the source value to the value at that index.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.action.step("triggerModelName", "blockName", channel)
slot[Z].trigger.model.addblock.source.action.step("triggerModelName", "blockName",
{channelList})
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block defining where to branch; an empty string is a valid block name
channel	Channel number; see **Details**
channelList	Table of channels to take measurements on; for example: `{channel1, channel2,...}`. See **Details**.

Details

To create pulses that go to the voltage in the sweep array and then to the bias voltage, you can use this command with the slot[Z].trigger.model.addblock.source.action.bias() (on page 600) command.

This command advances the source output for the specified channel to the next value in the pre-defined sweep list. The sweep list must be created before running the trigger model using one of the supported sweep configuration commands.

- `slot[Z].smu[X].trigger.source.linearY`
- `slot[Z].smu[X].trigger.source.listY`
- `slot[Z].smu[X].trigger.source.logY`
- `slot[Z].psu[X].trigger.source.linearY`
- `slot[Z].psu[X].trigger.source.listY`
- `slot[Z].psu[X].trigger.source.logY`

Each time the block is executed, the sweep index increases by one, updating the output to the corresponding sweep value. Once the end of the sweep list is reached, it starts again at the beginning.

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

The channels are specified as either integers or slot notation `ZOOX`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

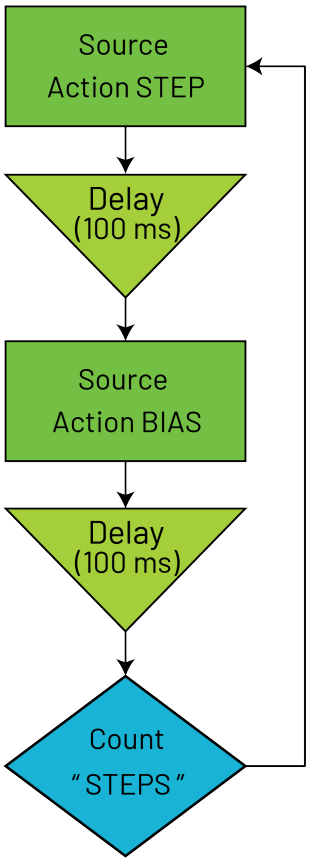
All specified channels must be on the module running the trigger model.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.linearv(0, 5, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "step", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.action.bias("myTM", "", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "step", 10)
-- Turn on the output before the trigger model is initiated.
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model that increases from 0 V to 5 V with 5 steps at 1 V for each step.

Trigger model diagram



Also see

slot[Z].trigger.model.addblock.source.action.bias() (on page 600)

slot[Z].trigger.model.addblock.source.action.skip() (on page 608)
Source action step block (on page 73)

Sweep configuration commands

slot[Z].psu[X].trigger.source.linearY (on page 405)
slot[Z].smu[X].trigger.source.linearY (on page 532)
slot[Z].psu[X].trigger.source.listY (on page 406)
slot[Z].smu[X].trigger.source.listY (on page 533)
slot[Z].psu[X].trigger.source.logY (on page 408)
slot[Z].smu[X].trigger.source.logY (on page 535)

slot[Z].trigger.model.addblock.source.output()

This function defines a trigger model block that turns the output source on or off.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.source.output("triggerModelName", "blockName", channel, state)
slot[Z].trigger.model.addblock.source.output("triggerModelName", "blockName", {channelList}, state)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block to be added; an empty string is a valid block name
channel	Module channel number; see **Details**
channelList	Table of channels to make measurements on; for example: `{channel1, channel2,...}`; see **Details**
state	**PSUs** Turn the source off: `slot[Z].psu.OFF` Turn the source on: `slot[Z].psu.ON` **SMUs** Turn the source off: `slot[Z].smu.OFF` Turn the source on: `slot[Z].smu.ON`

Details

The source output block determines if the output source is turned on or off when the trigger model reaches this block. This block does not determine the settings of the output source, such as the output voltage level. The source settings are determined by either the present settings of the instrument or by a configuration list.

Although pulses can be created by turning the output on and off, the instrument typically performs multiple actions when turning the output on and off. For more precise timing, use the slot[Z].trigger.model.addblock.source.action.step() (on page 610) and slot[Z].trigger.model.addblock.source.action.bias() (on page 600) blocks to create pulses at the output.

You can specify one or more channels for the source action to run on using *channelList*. All source actions run on all specified channels in parallel, plus or minus 500 ns. The trigger model then waits for all measurements to complete before continuing.

The channels are specified as either integers or slot notation `ZOOX`, where `Z` is the slot number and `X` is the channel number. For example, channel 1 on slot 1 is specified as either `1` or `1001`.

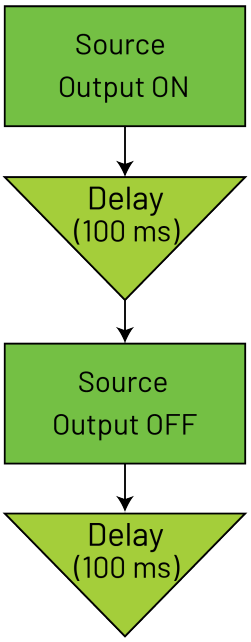
All specified channels must be on the module running the trigger model.

Example

```
-- Configure the source level.
slot[1].psu[1].source.levelv = 5.0
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].ON)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].OFF)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
```

This example demonstrates how to use the source.output block in a trigger model to turn the output on and off with a delay.

Trigger model diagram



Also see

Source output block (on page 74)

slot[Z].trigger.model.addblock.wait()

This function defines a trigger model block that waits for up to four events before allowing the trigger model to continue.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.addblock.wait("triggerModelName", "blockName", event)
slot[Z].trigger.model.addblock.wait("triggerModelName", "blockName", event, clear)
slot[Z].trigger.model.addblock.wait("triggerModelName", "blockName", event, clear, logic, event)
slot[Z].trigger.model.addblock.wait("triggerModelName", "blockName", event, clear, logic, event,
    event)
slot[Z].trigger.model.addblock.wait("triggerModelName", "blockName", event, clear, logic, event,
    event, event)
```

Z	Module slot number
triggerModelName	Name of the trigger model to which the block is added
blockName	Name of the trigger block being added by this command; an empty string is a valid block name
event	The event that must occur before the trigger block allows trigger execution to continue (see **Details**)
clear	Determines how previously detected events are handled when entering the wait block: ▪ Clear events: `slot[Z].trigger.model.CLEAR_ENTER` ▪ Retain events: `slot[Z].trigger.model.CLEAR_NEVER`
logic	Specifies how multiple events are evaluated: ▪ Continue the trigger model when all listed events have occurred: `slot[Z].trigger.model.WAIT_AND` ▪ Continue the trigger model when any listed event occurs: `slot[Z].trigger.model.WAIT_OR`

Details

You can use this command to synchronize measurements with other instruments and devices. You can set the instrument to wait for the following events:

- Notify
- Command interface trigger
- Digital input/output signals, such as digio and TSP-Link

The event can occur before trigger model execution reaches the wait block. If the event occurs after trigger model execution starts but before the trigger model execution reaches the wait block, the trigger model records the event.

When *clear* is set to `CLEAR_ENTER`, the wait responds only to events that occur after the trigger model enters the block. When it is set to `CLEAR_NEVER`, if required events already occurred, the trigger model continues immediately. For both options, all events are cleared when the wait block completes.

You can use the event IDs in the following table with this command.

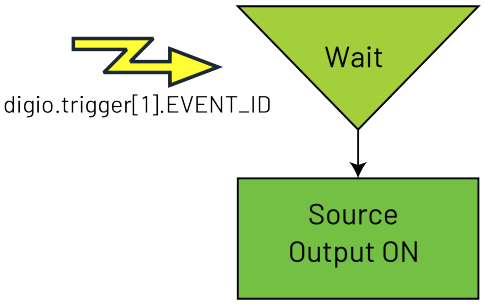
ID	Description
digio.trigger[N].EVENT_ID (on page 146)	Occurs when a digital I/O line changes state according to its configured trigger mode; *N* is 1 to 18
slot[Z].smu[X].trigger.EVENT_AT_LIMIT (on page 529) (SMUs only)	Occurs when the channel of a SMU limits the output to stay within user-defined constraints
slot[Z].trigger.model.EVENT_NOTIFYN (on page 618)	Occurs when the notify block of the trigger model is executed; *N* is 1 to 8 for SMUs and 1 to 16 for PSUs
trigger.blender[N].EVENT_ID (on page 299)	Trigger blender event. Occurs when one or more combined events occur; *N* is the blender number, 1 to 4
trigger.generator[N].EVENT_ID (on page 308)	Trigger generator event; *N* is the generator number, 1 to 8
trigger.timer[N].EVENT_ID (on page 310)	Occurs when the timer delay of timer *N* elapses; *N* is 1 to 8
tsplink.trigger[N].EVENT_ID (on page 322)	Occurs when a TSP-Link line changes state according to its configured trigger mode; *N* is 1 to 3

Example

```
-- Configure the digital I/O trigger.
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
digio.trigger[1].edge = digio.EDGE_RISING
-- Configure the source level.
slot[1].psu[1].source.levelv = 5
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.wait("myTM", "", digio.trigger[1].EVENT_ID)
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].ON)
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Assert the digio trigger event to notify the trigger model.
digio.trigger[1].assert()
-- Wait for the trigger model to complete.
waitcomplete()
-- Turn the source output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example waits for a digital I/O trigger event to occur before turning on the output of a channel.

Trigger model diagram



Also see

Wait block (on page 70)

slot[Z].trigger.model.create()

This function creates a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.create("triggerModelName")`

Z	Module slot number
triggerModelName	Name of the trigger model to create

Details

You can create multiple trigger models on a module. Each trigger model must have a unique name and must not be an empty string.

You can add trigger model blocks with the `slot[Z].trigger.model.addblock.*` series of functions. You must reference the name of the trigger model in each `addblock` command.

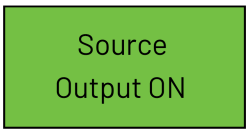
To start the trigger model, use `slot[z].trigger.model.initiate("triggerModelName")`.

Example

```
-- Configure the source level.
slot[1].psu[1].source.levelv = 5
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].ON)
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for trigger models to complete.
waitcomplete()
-- Turn the output off.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example creates a trigger model, adds a block, starts the trigger model, and then waits for it to complete.

Trigger model diagram



Also see

slot[Z].trigger.model.delete() (on page 617)
slot[Z].trigger.model.initiate() (on page 619)

slot[Z].trigger.model.delete()

This function deletes a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.delete("triggerModelName")`

Z	Module slot number
triggerModelName	Name of the trigger model to delete

Details

This function deletes a trigger model. If a trigger model has been loaded, it will be unloaded as a part of the deletion process. Attempting to delete a running trigger model will cause an error.

You can remove trigger model blocks with `slot[Z].trigger.model.removeblock()`.

Example

```
slot[1].trigger.model.create("delete1")
slot[1].trigger.model.delete("delete1")
```

Creates and then deletes a trigger model named `delete1` on the module installed in slot 1.

Also see

slot[Z].trigger.model.create() (on page 616)
slot[Z].trigger.model.initiate() (on page 619)
slot[Z].trigger.model.load() (on page 620)
slot[Z].trigger.model.removeblock() (on page 621)

slot[Z].trigger.model.EVENT_NOTIFYN

This constant contains the trigger model notify event number.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Constant	Yes			

Usage

```
eventId = slot[Z].trigger.model.EVENT_NOTIFYN
```

eventId	The trigger event number
Z	Module slot number
N	The notify event number; see **Details**

Details

Use this constant to specify the event generated by a trigger model notify block or have other trigger objects respond to the event generated by the notify block.

To have another trigger object respond to trigger events generated by this notify block, set the stimulus of the other object to the value of this constant.

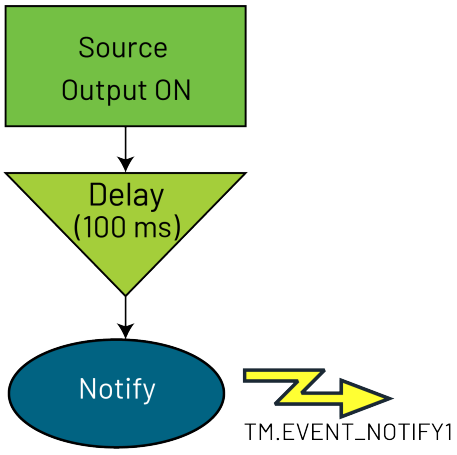
Valid notify event numbers (*N*) for PSU modules are from 1 to 16. For SMU modules, from 1 to 8.

Example

```
digio.trigger[1].mode = digio.MODE_TRIGGER_OUT
digio.trigger[1].stimulus = slot[1].trigger.model.EVENT_NOTIFY1
```

Sets digital I/O line 1 to be an output trigger line and configures the stimulus to be `NOTIFY1` on slot 1. When `EVENT_NOTIFY1` is encountered in the trigger model, digital I/O line 1 changes state.

Trigger model diagram



Also see

Notify blocks (on page 74)
slot[Z].trigger.model.addblock.notify() (on page 596)

slot[Z].trigger.model.initiate()

This function starts a specified trigger model on a module.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

```
slot[Z].trigger.model.initiate("triggerModelName")
```

Z	Module slot number
triggerModelName	Name of the trigger model to start

Details

This command starts the named trigger model. The trigger model will begin to run in the background by the end of this function.

If any trigger model blocks are improperly configured, or if the trigger model is already running or has not been created, the trigger model will not start, and an error message will be returned.

Sending slot[Z].trigger.model.load() (on page 620) is not necessary for this command to execute a trigger model.

If the trigger model is loaded using slot[Z].trigger.model.load() (on page 620), this command executes without performing the load step, and the trigger model remains loaded after completion. If the trigger model is not loaded, this command loads it before execution and unloads it when the run completes.

If the trigger model is loaded, the instrument also restores all channels used by that model to the pre-run, pre-execution state. This ensures consistent and repeatable operation across multiple runs.

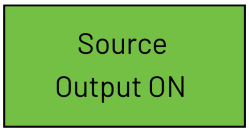
To pause execution until a trigger model is complete, use waitcomplete() (on page 347). To abort a running trigger model, use slot[Z].trigger.model.abort() (on page 570).

Example

```
slot[1].psu[1].source.levelv = 5
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].ON)
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for trigger models to complete.
waitcomplete()
-- Turn off the source output.
slot[1].psu[1].source.output = slot[1].psu[1].OFF
```

This example creates a trigger model, adds a block, starts the trigger model, and then waits for it to complete.

Trigger model diagram



Also see

slot[Z].trigger.model.abort() (on page 570)
slot[Z].trigger.model.load() (on page 620)

slot[Z].trigger.model.load()

This function loads a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.load("triggerModelName")`

Z	Module slot number
triggerModelName	Name of the trigger model to load

Details

This function loads a previously-created trigger model into the channels specified in the trigger model and prepares it for execution. Loading a trigger model configures the internal execution state so that a subsequent call to `slot[Z].trigger.model.initiate` can begin execution more quickly.

When a trigger model is loaded, the channel is placed into a protected state. Query and getter commands are permitted, but configuration changes that alter the channel's operating mode, source, measure, or range settings are not permitted after the trigger model is loaded, with limited exceptions.

For example:

- **Permitted**: `print(slot[1].psu[2].source.levelv)`
- **Not permitted:** `slot[1].psu[2].source.levelv = 3.5`

The following exceptions are supported for checking the contacts of a DUT after a trigger model is loaded:

- `slot[Z].smu[X].contact.check()`
- `slot[Z].smu[X].contact.r()`
- `slot[Z].smu[X].source.limiti()`
- `slot[Z].smu[X].source.offlimiti()`
- `slot[Z].smu[X].source.rangei()`

After a trigger model is loaded, the trigger model definition cannot be modified. Commands that add, remove, or edit trigger model blocks are rejected until the trigger model is unloaded.

When `slot[Z].trigger.model.load()` is called, the source and measure configurations for each channel used in the trigger model are captured. If the trigger model changes channel configuration during execution, those changes remain until the trigger model completes. The pre-load state is then restored as the starting conditions for subsequent `slot[Z].trigger.model.initiate()` calls.

The pre-load state includes the channel output state. For example, if a channel's output is turned on when a trigger model is loaded and the output is turned off in the trigger model, the output is turned back on after the trigger model has completed.

Loading a trigger model does not begin execution. Send `slot[Z].trigger.model.initiate()` to begin execution.

To remove the protected state and allow configuration or edits to the trigger model, send `slot[Z].trigger.model.unload()`.

Deleting or aborting a trigger model (from inside or outside of the trigger model) also unloads it and removes the protected state.

Example

```
-- Configure the sweep.
slot[1].smu[1].trigger.source.linearv(0, 5, 10)
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "step", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "step", 10)
-- Turn on the output before the trigger model is loaded.
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Load the trigger model
slot[1].trigger.model.load("myTM")
-- Initiate the trigger model. This initiate will begin
-- with reduced latency compared to a non-loaded trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
-- The source level returns to 0 V and the output is turned on.
-- Initiate the trigger model again. This initiate will begin
-- with reduced latency compared to a non-loaded trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete
waitcomplete()
-- Unload the trigger model
slot[1].trigger.model.unload("myTM")
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model loaded before being executed twice and then unloaded once it is complete. Initiates will begin with reduced latency compared to non-loaded trigger models.

Also see

Load or unload a trigger model (on page 88)
slot[Z].trigger.model.initiate() (on page 619)
slot[Z].trigger.model.state() (on page 623)
slot.[Z].trigger.model.unload() (on page 624)

slot[Z].trigger.model.removeblock()

This function removes every trigger block with the specified name from the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.removeblock("triggerModelName", "triggerBlockName")`

Z	Module slot number
triggerModelName	Name of the trigger model from which the trigger blocks will be removed
triggerBlockName	Name of the trigger block to remove

Details

When this command is executed, any block with the specified trigger block name (including the empty string) in the named trigger model is removed. If either the trigger block name or the trigger model name does not exist, an error message is returned.

You can add trigger model blocks with the `slot[Z].trigger.model.addblock.*` series of functions.

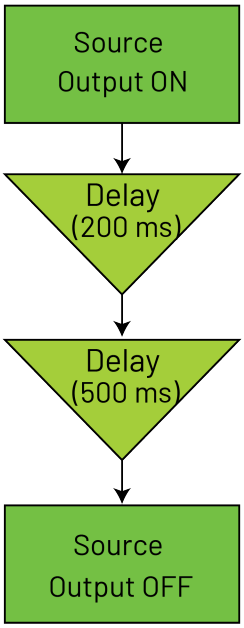
Example

```
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].ON)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.2)
slot[1].trigger.model.addblock.delay.constant("myTM", "Delay2", 0.5)
slot[1].trigger.model.addblock.source.output("myTM", "", 1, slot[1].psu[1].OFF)
-- Remove the second delay block.
slot[1].trigger.model.removeblock("myTM", "Delay2")
-- Initiate the trigger model.
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete.
waitcomplete()
```

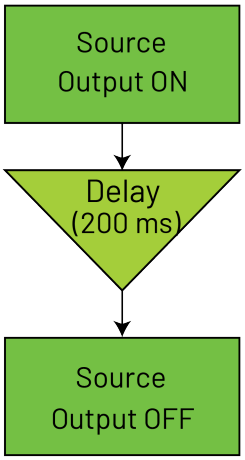
This example uses the `slot[Z].trigger.model.removeblock()` command to delete an existing trigger model block.

Trigger model diagram

myTM before remove block command



myTM after remove block command



Also see

slot[Z].trigger.model.create() (on page 616)
slot[Z].trigger.model.delete() (on page 617)
slot[Z].trigger.model.initiate() (on page 619)

slot[Z].trigger.model.state()

This function returns the present state of the trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

status = slot[*Z*].trigger.model.state(*"triggerModelName"*)

status	The status of the trigger model: ▪ slot[*Z*].trigger.model.STATE_ABORTED ▪ slot[*Z*].trigger.model.STATE_ABORTING ▪ slot[*Z*].trigger.model.STATE_BUILDING ▪ slot[*Z*].trigger.model.STATE_EMPTY ▪ slot[*Z*].trigger.model.STATE_FAILED ▪ slot[*Z*].trigger.model.STATE_IDLE ▪ slot[*Z*].trigger.model.STATE_LOADED ▪ slot[*Z*].trigger.model.STATE_RUNNING
Z	Module slot number
triggerModelName	Name of the trigger model

Details

This function returns the state of the trigger model. The instrument checks the state of a trigger model as the model executes. A trigger model name is required; an empty set of parentheses will cause an error.

The trigger model states are:

- **Aborted:** The trigger model is stopped
- **Aborting:** The trigger model is stopping
- **Building:** The trigger model is created but has not run since new blocks were added
- **Empty:** The trigger model is created, but no blocks are defined
- **Failed:** The trigger model is stopped because of an error
- **Idle:** The trigger model has completed and is now stopped
- **Loaded:** The trigger model is loaded and has reserved hardware resources for rapid initiates
- **Running:** The trigger model is running

Example

```
-- Create the trigger model.
slot[1].trigger.model.create("myTM")
-- Print the state of the trigger model.
print(slot[1].trigger.model.state("myTM"))
-- Add a delay constant block to the trigger model.
slot[1].trigger.model.addblock.delay.constant("myTM", "delay", 0.1)
-- Print the state of the trigger model again.
print(slot[1].trigger.model.state("myTM"))
```

This example demonstrates how to create a trigger model, check its state, add a delay block, and check its state again.

After the trigger model is created, the state of the trigger model returns `STATE_EMPTY`. After the delay constant block is added, the state of the trigger model returns `STATE_BUILDING`.

Also see

slot[Z].trigger.model.abort() (on page 570)
slot[Z].trigger.model.initiate() (on page 619)
slot[Z].trigger.model.load() (on page 620)
slot[Z].trigger.model.unload() (on page 624)

slot[Z].trigger.model.unload()

This function unloads a trigger model.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Function	Yes			

Usage

`slot[Z].trigger.model.unload("triggerModelName")`

Z	Module slot number
triggerModelName	Name of the trigger model to unload

Details

This function unloads a previously-loaded trigger model and releases the protections for all channels used. After unloading, the channels can be reconfigured and the trigger model definition can be edited.

A trigger model is automatically unloaded if deleted or if a running trigger model is aborted.

Example

```
-- Configure the sweep
slot[1].smu[1].trigger.source.linearv(0, 5, 10)
-- Create the trigger model
slot[1].trigger.model.create("myTM")
slot[1].trigger.model.addblock.source.action.step("myTM", "step", 1)
slot[1].trigger.model.addblock.delay.constant("myTM", "", 0.1)
slot[1].trigger.model.addblock.branch.counter("myTM", "", "step", 10)
-- Turn on the output before the trigger model is loaded
slot[1].smu[1].source.levelv = 0
slot[1].smu[1].source.output = slot[1].smu[1].ON
-- Load the trigger model
slot[1].trigger.model.load("myTM")
-- Initiate the trigger model
slot[1].trigger.model.initiate("myTM")
-- Wait for the trigger model to complete
waitcomplete()
-- Unload the trigger model
slot[1].trigger.model.unload("myTM")
-- Turn off the output.
slot[1].smu[1].source.output = slot[1].smu[1].OFF
```

This example demonstrates a trigger model being loaded before being executed and then unloaded once it completes

Also see

Load or unload a trigger model (on page 88)
slot[Z].trigger.model.abort() (on page 570)
slot[Z].trigger.model.initiate() (on page 619)
slot[Z].trigger.model.state() (on page 623)
slot.[Z].trigger.model.unload() (on page 624)

Common commands

Common commands

The IEEE Std 488.2 common commands that are supported by the MP5000 Series are described in the following topics. Although commands are shown in uppercase, common commands are not case sensitive, so you can use either uppercase or lowercase. Although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, the MP5103 and supported modules do not strictly conform to that standard.

NOTE: Unlike other commands, like those listed in TSP commands, each common command must be sent in a separate message.

The common commands cannot be used in scripts.

The TSP commands that can be included in scripts that are equivalent to the common commands are defined in the following table.

Common command	Script command equivalent
`*CLS`	`status.reset()`
`*ESE?`	`print(tostring(status.standard.enable))`
`*ESE` *mask*	`status.standard.enable = mask`
`*ESR?`	`print(tostring(status.standard.event))`
`*IDN?`	`print(localnode.manufacturer..[[,]]..localnode.model..[[,]]..localnode.serialno.. [[,]]..localnode.revision)`
`*OPC?`	`waitcomplete() print([[1]])`
`*OPC`	`opc()`
`*RST`	`reset()`
`*SRE?`	`print(tostring(status.request_enable))`
`*SRE` *mask*	`status.request_enable = mask`
`*STB?`	`print(tostring(status.condition))`
`*TRG`	Not available
`*TST?`	`print([[0]])`
`*WAI`	`waitcomplete()`

*CLS

This command clears the event registers and queues in the mainframe.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*CLS
```

Details

Use the `*CLS` command to clear (reset to 0) the bits of the following registers in the MP5103:

- Operation Event register
- Event log
- Questionable Event register

It does not affect the Standard Event, the Questionable Event Enable, or the Operation Event Enable registers.

Example

```
*CLS
```

Clear the event registers and queues.

Also see

*ESE (on page 627)
eventlog.clear() (on page 159)
status.reset() (on page 283)

*ESE

This command sets and queries bits in the Status Enable register of the Standard Event register.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	See **Details**

Usage

```
*ESE value
*ESE?
```

value	The value of the Status Enable register of the Standard Event register: 0 to 255

Details

When a bit in the Status Enable register is set on and the corresponding bit in the Standard Event Status register is set on, the ESB bit of the Status Byte register is set to on.

To set a bit on, send the constant or the value of the bit as the `<n>` parameter.

Tou can set the bit as a constant or a numeric value, as shown in the following table. To set more than one bit of the register, you can send multiple constants with `+` between them. You can also set *standardRegister* to the sum of their decimal weights. For example, to set bits B0 and B2, set *standardRegister* to 5 (which is the sum of 1 + 4). You can also send:

```
status.standard.enable = status.standard.OPC + status.standard.QYE
```

When zero (0) is returned, no bits are set. You can also send `0` to clear all bits.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC command or TSP opc() (on page 218) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
3	8	Not used	Not used.
4	16	Not used	Not used.
5	32	Undefined	GET error: The instrument received a Group Execute Trigger (GET) inside a program message.
6	64	Not used	Not used.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Example

`*ESE 129`	`*ESE 129` sets the Status Enable register of the Standard Event register to binary `10000001`, which enables the PON and OPC bits.

Also see

*CLS (on page 626)
Standard Event Register (on page 635)
status.standard.* (on page 284)
Status model (on page 634)

*ESR?

This command reads and clears the contents of the Standard Event Status Register.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*ESR?
```

Details

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register and clears the Standard Event Status Register.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
0	1	`status.standard.OPC`	All pending selected instrument operations are complete and the instrument is ready to accept new commands. The bit is set in response to an *OPC command or TSP opc() (on page 218) function.
1	2	Not used	Not used.
2	4	`status.standard.QYE`	Attempt to read data from an empty Output Queue.
3	8	Not used	Not used.

Bit	Decimal value	Constant	When set, indicates the following has occurred:
4	16	Not used	Not used.
5	32	Undefined	GET error: The instrument received a Group Execute Trigger (GET) inside a program message.
6	64	Not used	Not used.
7	128	`status.standard.PON`	The instrument has been turned off and turned back on since the last time this register was read.

Example

```
*ESR?
```

Example output:

```
128
```

Shows that the Standard Event Status Register contains binary `10000000`, which indicates that the instrument was rebooted since the last time this register was read.

Also see

Status model (on page 634)
status.standard.* (on page 284)

*IDN?

This command retrieves the identification string of the instrument.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*IDN?
```

Details

The identification string includes the manufacturer, model number, serial number, and firmware revision of the mainframe. The string is formatted as follows:

```
TEKTRONIX,nnnn,xxxxxxxx,yyyyyy
```

Where:

- *nnnn* is the model number
- *xxxxxxxx* is the serial number
- *yyyyyy* is the firmware revision

NOTE: To retrieve the identification string of a module, use `slot[Z].model`.

Example

```
*IDN?
```

Output:

```
TEKTRONIX,MP5103,00000000,1.0.0
```

Also see

None

*OPC

This command sets the operation complete (OPC) bit after all pending commands, including overlapped commands, have been executed.

Type	Affected by	Where saved	Default value
Command and query	Not applicable	Not applicable	Not applicable

Usage

```
*OPC
*OPC?
```

Details

When `*OPC` is sent, the OPC bit (bit 0) in the Status Event Status Register is set after all pending command operations have been executed. After all programmed operations are complete, the instrument returns to idle, at which time all pending commands (including `*OPC` and `*OPC?`) are executed. After the last pending command is executed, the OPC bit is set or an ASCII "1" is placed in the Output Queue. The Output Queue returns "1" when `*OPC?` is sent.

When the trigger model is executing, most sent commands are not executed. If a command cannot be processed, an error event message is generated in the event log.

Example

`*OPC?`	If all pending OPC operations are complete, the return is `1`.

Also see

opc() (on page 218)

*RST

This command resets the instrument settings to their default values and clears the reading buffers.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*RST
```

Details

Returns the mainframe and all modules to default settings, cancels all pending commands, and cancels the response to any previously received `*OPC` and `*OPC?` commands.

Also see

reset() (on page 226)

*SRE

This command sets or clears and queries the bits of Status Request Enable Register.

Type	Affected by	Where saved	Default value
Command and query		Not applicable	0

Usage

```
*SRE <n>
*SRE?
```

`<n>`	Clear the Status Request Enable Register: `0` Set the instrument for an SRQ interrupt: `32`

Details

This command sets or clears the individual bits of the Status Request Enable Register.

The Status Request Enable Register is cleared when power is cycled or when a parameter value of 0 is sent with this command.

The instrument returns a decimal value that corresponds to the binary-weighted sum of all bits set in the register.

Bit	Decimal value	Constants	When set, indicates the following has occurred:
0	1	`status.MSB`	An enabled event in the Measurement Event Register has occurred.
1	2	Not used	Not used.
2	4	`status.EAV`	An error or status message is present in the Error Queue.
3	8	`status.QSB`	An enabled event in the Questionable Status Register has occurred.
4	16	`status.MAV`	A response message is present in the Output Queue.
5	32	`status.ESB`	An enabled event in the Standard Event Status Register has occurred.
6	64	Not used	Not used.
7	128	`status.OSB`	An enabled event in the Operation Status Register has occurred.

Example

`*SRE 0`	Clear the bits of the Status Request Enable Register.

Also see

None

*STB?

This command gets the status byte of the instrument without clearing the request service bit.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	Not applicable

Usage

```
*STB?
```

Details

This command is similar to a serial poll, but it is processed like any other instrument command.

The `*STB?` command returns the same result as a serial poll, but the master summary bit (MSB) is not cleared if a serial poll has occurred. The MSB is not cleared until all other bits feeding into the MSB are cleared.

Example

`*STB?`	Queries the status byte.

Also see

None

*TRG

This command generates a trigger event from a remote command interface.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*TRG
```

Details

Use the `*TRG` command to generate a trigger event for `trigger.wait()`.

Also see

trigger.wait() (on page 314)

*TST?

This command is accepted and returns 0. A self-test is not actually performed.

Type	Affected by	Where saved	Default value
Query only	Not applicable	Not applicable	0

Usage

```
*TST?
```

Also see

None

*WAI

This command postpones the execution of subsequent commands until all previous overlapped commands are finished.

Type	Affected by	Where saved	Default value
Command only	Not applicable	Not applicable	Not applicable

Usage

```
*WAI
```

Details

There are two types of instrument commands:

- **Overlapped commands:** Commands that allow the execution of subsequent commands while instrument operations of the overlapped command are still in progress.
- **Sequential commands:** Commands whose operations must finish before the next command is executed.

The `*WAI` command suspends the execution of commands until the instrument operations of all previous overlapped commands are finished. The `*WAI` command is not needed for sequential commands. Typically, this command is sent after the initiate trigger model command.

Also see

waitcomplete() (on page 347)

Status model

MP5103 status model

Overview

The status model includes status registers and queues that allow you to monitor and control instrument events. You can include commands in your test program that can determine if a service request (SRQ) event has occurred and the cause of the event.

All status model registers and queues flow into the Status Byte register.

Status register set contents

Typically, a status register set contains the following registers: Condition, Enable, Event, Negative-transition (NTR), and Positive transition (PTR).

When an event occurs and the appropriate NTR or PTR bit is set, the corresponding event register bit is set to 1. The event bit remains latched to 1 until the Event register is read or the status model is reset. When an Event register bit is set and its corresponding enable bit is set, the summary bit of the register is set to 1. This, in turn, sets a bit in a higher-level condition register, potentially cascading to the associated summary bit of the Status Byte Register.

Condition register

The Condition register is a read-only register that is constantly updated to reflect the present operating conditions of the instrument. An event is represented by a change in the Condition register bit from a 1 to 0 or 0 to 1.

This register is the `.condition` option of each attribute.

Enable register

The Enable Register is read/write and allows a summary bit of the register to be set when an enabled event occurs.

This register is the `.enable` option of each attribute.

Event register

The Event register is read-only and 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 is set to 1.

When an event occurs and the appropriate NTR or PTR bit is set, the corresponding event register bit is set to 1. The event bit remains latched to 1 until the Event register is read or the status model is reset

This register is the `.event` option of each attribute.

Negative transition register

The Negative Transition Register (NTR) is read/write. When a bit is set in this register, it enables a 1 to 0 change in the corresponding bit of the Condition register to cause the corresponding bit in the Event register to be set.

This register is the `.ntr` option of each attribute.

Positive transition register

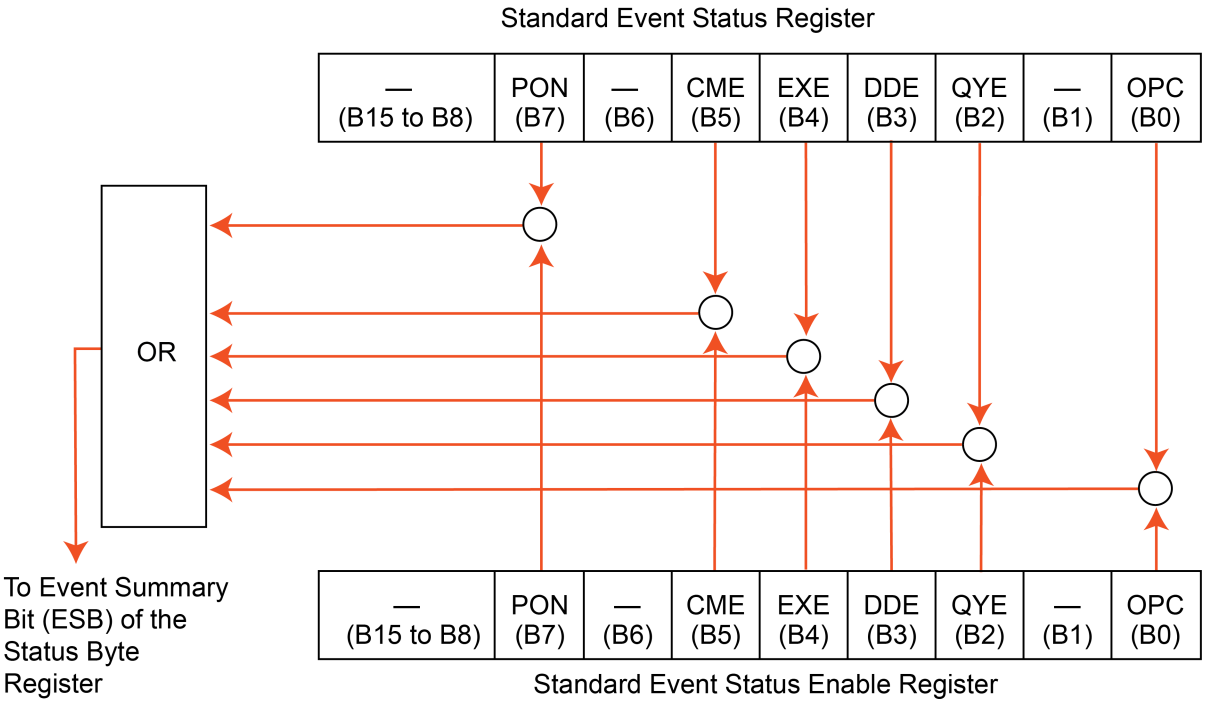
The Positive Transition Register (PTR) is read/write. When a bit is set in this register, it enables a 0 to 1 change in the corresponding bit of the condition register to cause the corresponding bit in the event register to be set.

This register is the `.ptr` option of each attribute.

Standard Event Register

The Standard Event Register set includes two 8-bit registers:

- **Standard Event Status Register:** Reports when a predefined event has occurred. The register latches the event and the corresponding bit remains set until it is cleared by a read.
- **Standard Event Status Enable Register:** You can enable or disable bits in this register. This allows the predefined event (from the Standard Event Status Register) to set the ESB of the Status Byte Register.



The bits in the Standard Event Register are:

- **Bit B0, Operation Complete (OPC):** Set bit indicates that all pending selected device operations are completed and the MP5000 Series instrument is ready to accept new commands. The bit is set in response to an *OPC command. The `opc()` function can be used in place of the *OPC command. See Common commands for details on the *OPC command.
- **Bit B1:** Not used.
- **Bit B2, Query Error (QYE):** Set bit indicates that you attempted to read data from an empty output queue.
- **Bit B3, Device-Dependent Error (DDE):** Set bit indicates that an instrument operation did not execute properly due to some internal condition.
- **Bit B4, Execution Error (EXE):** Set bit indicates that the MP5000 Series instrument detected an error while trying to execute a command.
- **Bit B5, Command Error (CME):** Set bit indicates that a command error has occurred. Command errors include:
  - IEEE Std 488.2 syntax error: The MP5000 Series instrument received a message that does not follow the defined syntax of IEEE Std 488.2.
  - Semantic error: MP5000 Series instrument received a command that was misspelled or received an optional IEEE Std 488.2 command that is not implemented.
- **Bit B7, Power ON (PON):** Set bit indicates that the MP5000 Series instrument has been turned off and turned back on since the last time this register was read.

Commands to program and read the register are summarized in the following table.

Standard event commands	
Commands	**Description**

`*ESR?`
or
`print(status.standard.event)`
	Read the Standard Event Status Register.

`*ESE mask`
or
`status.standard.enable = mask`
	Program the Event Status Enable Register:
	mask = 0 to 255

`*ESE?`
or
`print(status.standard.enable)`
	Read Event Status Enable Register.

For additional information, refer to Status register set contents (on page 634) and the following command descriptions:

- status.standard.* (on page 284)
- *ESR? (on page 628)
- *ESE (on page 627)

Program enable and transition registers

You can program the Enable and Transition registers of the status model registers. All other registers in the status structure are read-only.

When you program an Enable register bit to 0, no action occurs if the bits in the corresponding registers are set to 1.

When you program an Enable register bit to 1, if the bits in the corresponding registers are set to 1, the AND condition occurs and the summary bit in the Status Byte Register is set to 1.

You must program all bits in an enable register at the same time. To do this, determine what each bit value in the register will be, then add them together to determine the value of all the bits in the register.

For example, you might want to enable the Standard Event Register to set the ESB bit in the Status Byte Register whenever an operation complete occurs or whenever an operation did not execute properly because of an internal condition. To do this, you need to set bits 0 and 3 of the Standard Event Register to 1. These bits have decimal values of 1 and 8, so to set both bits to 1, you set the register to 9.

Send the command:

```
status.standard.enable = 9
```

Alternatively, you can program bits in the register using constants. For example:

```
status.standard.enable = status.standard.OPC + status.standard.DDE
```

A command to program an Enable or Transition register is sent with a parameter value that determines the state (0 or 1) of each bit in the appropriate register. The bit positions of the register indicate the binary parameter value and decimal equivalent. To program one of the registers, send the decimal value for the bits to be set. The bit position, binary value, decimal value, and bit weight of each bit is shown in the following tables.

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

Bit	B7	B6	B5	B4	B3	B2	B1	B0
Decimal	128	64	32	16	8	4	2	1
Weights	(2^7)	(2^6)	(2^5)	(2^4)	(2^3)	(2^2)	(2^1)	(2^0)
Bit	**B15**	**B14**	**B13**	**B12**	**B11**	**B10**	**B9**	**B8**
Binary value	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1
Decimal	32,768	16,384	8,192	4,096	2,048	1,024	512	256
Weights	(2^{15})	(2^{14})	(2^{13})	(2^{12})	(2^{11})	(2^{10})	(2^9)	(2^8)

When using a numeric parameter, registers are programmed by adding the values of the bits. For example:

```
*ese 1169
status.standard.enable = 1169
```

To convert from decimal to binary, use the information shown in the previous table. For example, to set bits B0, B4, B7, and B10, use a decimal value of 1169 for the command parameter (1169 = 1 + 16 + 128 + 1024).

Read the registers

You can read any register in the status model. The response is a decimal value that indicates which bits in the register are set, as described in Program enable and transition registers (on page 636).

To read the register, print the TSP command. You can use either `print()`, which returns the decimal value, or `print(tostring())`, which returns the string equivalent of the decimal value. You can also use common commands to read the register.

For example, you can send any one of the following commands to read the Status Enable Register of the Standard Event Register:

```
print(status.standard.enable)
*ese?
print(tostring(status.standard.enable))
```

Reset and clear registers

Registers in the status model can be cleared using commands or by instrument actions. When a register is cleared, the bits in the register are set to 0.

The event log and all registers are cleared when instrument power is cycled.

When you read a bit from the Operation Event, Questionable Event, or Standard Event Status Register, the entire 16-bit or 8-bit register value is returned and the event register is cleared or set to 0.

You can use the following commands to reset the registers:

- `*CLS`: This command also clears the output queue.
- `status.reset()`: Resets the mainframe status model.
- `slot[Z].status.reset()`: Resets the status model of the module in the defined slot.

When a reset command is sent, it resets the bits of the event and NTR registers to 0 and sets all PTR register bits to on.

You can also programmatically set the Enable registers and the NTR to 0. To do this, send the individual command to program the register with a `0` as its parameter value. You can reset the PTR registers to their defaults by programming them with all bits on.

The Event registers are not programmable, but you can clear them by reading them.

Status byte and service request (SRQ)

Service requests (SRQs) allow an instrument to indicate that it needs attention or that some event has occurred. When the controller receives an SRQ, it allows the controller to interrupt tasks to perform other tasks in order to address the request for service.

For example, you might program your instrument to send an SRQ when:

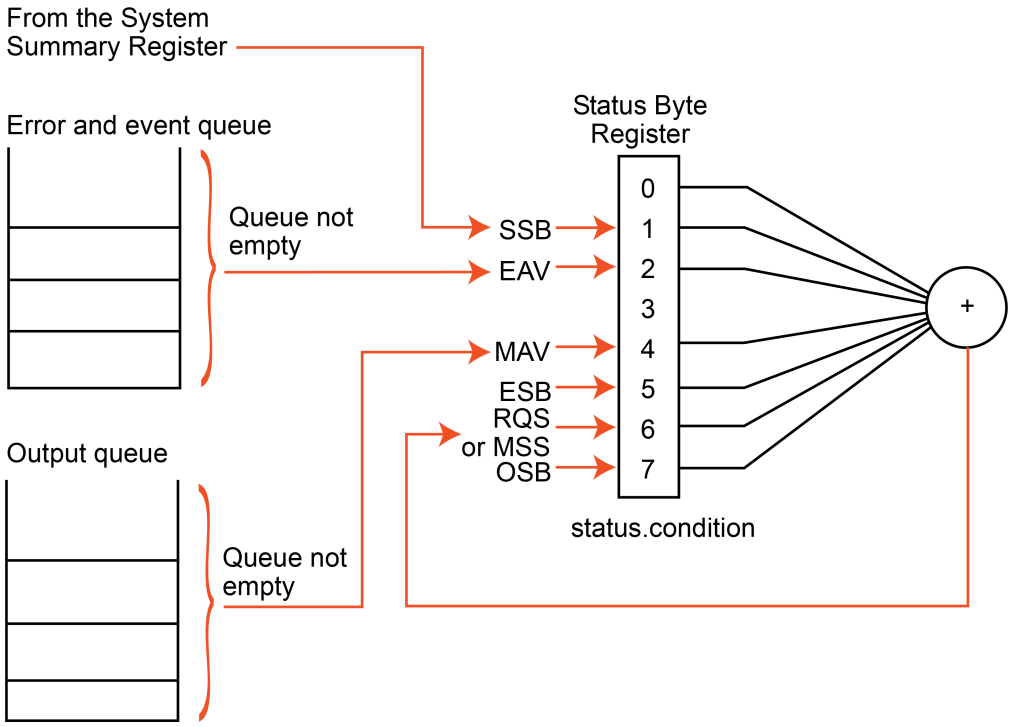
- All instrument operations are complete
- An instrument error occurs
- A specific operation has occurred

To control service requests, you use two 8-bit registers: The Status Byte Register and the Service Request Enable Register.

Service requests generate an SRQ event.

Status Byte Register

The Status Byte Register monitors the registers and queues in the status model and generates service requests (SRQs).



Bit name	Description
SSB	System Summary Bit
EAV	Error Available
MAV	Message Available
ESB	Event Summary Bit
RQS or MSS	Request Service or Master Summary Status
OSB	Operation Summary Bit

When bits are set in the status model registers and queues, they generate summary messages that set or clear bits of the Status Byte Register. You can enable these bits to generate an SRQ. 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 is cleared. As a result, its summary message resets to 0, which then resets the ESB bit in the Status Byte Register.

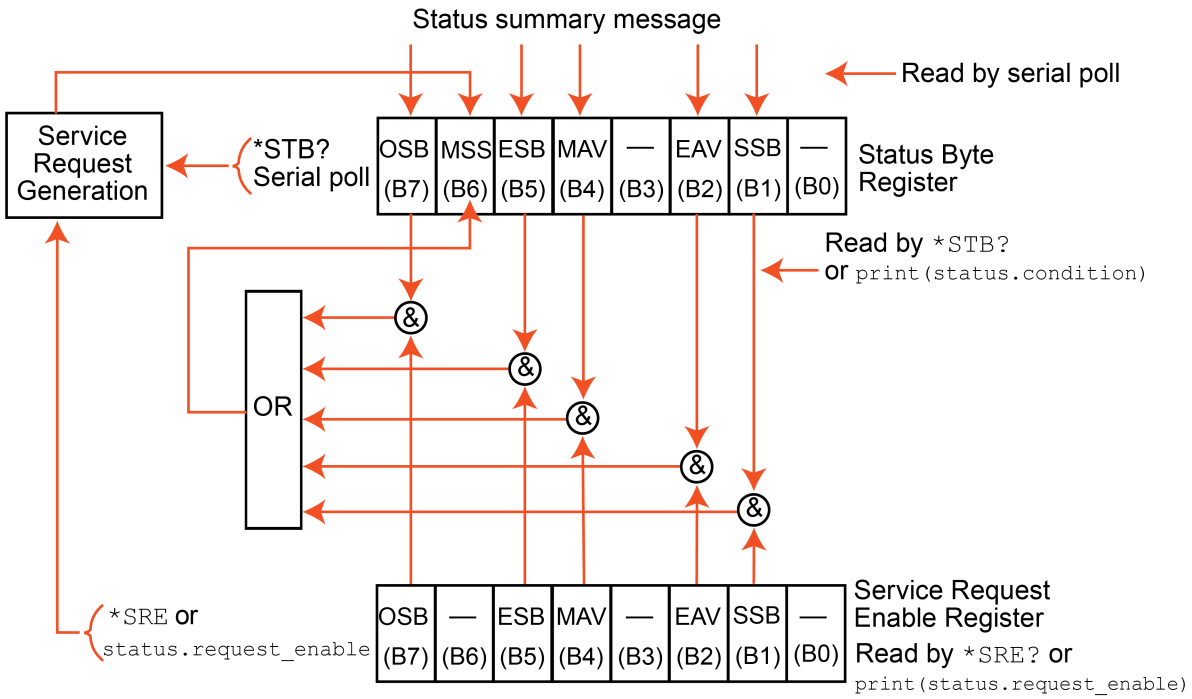
Service requests (SRQs) instruct the controller that the instrument needs attention or that some event has occurred. When the controller receives an SRQ, the controller can interrupt existing tasks to perform tasks that address the request for service.

The Status Byte Register also receives summary bits from itself, which sets the Master Summary Status (MSS) bit.

To reset the bits of the Service Request Enable Register to 0, use 0 as the parameter value for the command (for example, `*SRE 0` or `status.request_enable = 0`).

You can read and set which bits to AND in the Status Byte Register using the following commands.

Description	Common command	TSP command
Read the Status Byte Register	*STB? (on page 632)	status.condition (on page 237)
Read the Status Request Enable Register	*SRE (on page 631)	status.request_enable (on page 279)
Enable bits in the Status Request Enable Register	*SRE (on page 631)	status.request_enable (on page 279)



The bits of the Status Byte Register are described as follows:

- **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 event log.
- **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 USB serial poll sequence of the MP5000 Series to obtain the status byte (serial poll byte), B6 is the RQS bit. See Serial polling and SRQ (on page 640), Serial polling and SRQ (on page 642) for details on using the serial poll sequence.
  - When using the `*STB?` common command or `status.condition` Status byte and service request commands to read the status byte, B6 is the MSS bit.
- **Bit B7, Operation Summary (OSB):** Set summary bit indicates that an enabled operation event has occurred.

Service Request Enable Register

The 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, a logical AND operation is performed on the summary bits (&) with the corresponding enable bits of the Service Request Enable Register. When a logical AND operation is performed with a set summary bit (1) and with an enabled bit (1) of the enable register, the logic “1” output is applied to the input of the logical OR gate and, therefore, sets the MSS/RQS bit in the Status Byte Register.

You can set or clear the individual bits of the Service Request Enable Register by using the `*SRE` common command or `status.request_enable`. To read the Service Request Enable Register, use the `*SRE?` query or `print(status.request_enable)`. The Service Request Enable Register clears when power is cycled or a parameter value of 0 is sent with a status request enable command (for example, a `*SRE 0` or `status.request_enable = 0` is sent). You can program and read the SRQ Enable Register using the following commands.

Description	Common command	TSP command
Read the Status Request Enable Register	*SRE (on page 631)	status.request_enable (on page 279)
Enable bits in the Status Request Enable Register	*SRE (on page 631)	status.request_enable (on page 279)

Serial polling and SRQ

Any enabled event summary bit that transitions from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that services the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to Reset and clear registers (on page 637).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

Queues

The MP5103 uses queues to store messages. The queues include:

- **Command queue:** Holds commands that are available for execution.
- **Output queue:** Holds response messages.
- **Error and event queue:** Holds error, event, and status messages.

When a queue contains data, it sets the condition bit for that queue in one of the registers. The condition bits are:

- **Command queue:** CAV in the Operation Status Remote Summary Register.
- **Output queue:** MAV in the Status Byte Register.
- **Error and event queue:** EAV in the Status Byte Register.

The CAV, MAV, and EAV bits in the registers are cleared when the queue is empty. Queues empty when:

- Commands are executed.
- Errors and events are read from the error and event queue.
- Response messages are read from the instrument.

All queues are first-in, first-out (FIFO). However, if the error and event queue overflows, the newest event is discarded. An indicator message that the queue overflowed is retained.

Command queue

The command queue holds commands that have been received from a remote interface that are available for execution. This allows the MP5103 to accept multiple commands and queue them for execution.

When a command is received from a remote interface, the command available (CAV) bit in the Operation Status Remote Summary Register is set. For additional detail, see status.operation.remote.* (on page 268).

Output queue

Response messages, such as those generated from print commands, are placed in the output queue. All remote command interfaces share the same output queue.

The output queue sets the message available (MAV) bit in the status model.

The data in the output queue is cleared by the `*CLS` command. The bit is cleared when the Output Queue is empty.

Error and Event queue

The Error and Event queue holds error and status messages. As programming errors and status messages occur, a message that defines the error or status is placed in the Error and Event queue.

An error or status message is cleared from the Error and Event queue when it is read. You can also clear the queue by sending the command `eventlog.clear()`. An empty queue clears the error available (EAV) bit in the Status Byte Register.

Messages in the Error and Event queue include a code number, message text, severity, and TSP-Link™ node number. See Event log for a list of the messages.

When you read a single message from the Error and Event queue, the oldest message is read. If you attempt to read the Error and Event queue when it is empty, the error number 0 and `Queue is Empty` is returned.

The commands used to control the Error and Event queue are listed in the following table.

Error and Event queue command	Description
eventlog.clear() (on page 159)	Clear the Error and Event queue.
eventlog.count (on page 159)	The number of messages in the Error and Event queue.
eventlog.next() (on page 160)	Reads and returns the oldest unread error or event message from the Error and Event queue. The message is removed from the queue.

Serial polling and SRQ

Any enabled event summary bit that transitions from 0 to 1 sets bit B6 and generates a service request (SRQ).

In your test program, you can periodically read the Status Byte to check if an SRQ occurred and what caused it. If an SRQ occurred, the program can, for example, branch to an appropriate subroutine that services the request.

SRQs can be managed by the serial poll sequence of the instrument. If an SRQ does not occur, bit B6 (RQS) of the Status Byte Register remains cleared, and the program proceeds normally after the serial poll is performed. If an SRQ does occur, bit B6 of the Status Byte Register is set, and the program can branch to a service subroutine when the SRQ is detected by the serial poll.

The serial poll automatically resets RQS of the Status Byte Register. This allows subsequent serial polls to monitor bit B6 for an SRQ occurrence that is generated by other event types.

The serial poll does not clear the low-level registers that caused the SRQ to occur. You must clear the low-level registers explicitly. Refer to Reset and clear registers (on page 637).

For common commands and TSP commands, B6 is the MSS (Message Summary Status) bit. The serial poll does not clear the MSS bit. The MSS bit remains set until all enabled Status Byte Register summary bits are reset.

Status model configuration example

In this example, a current limit (compliance) event in the PSU module in slot 2 of node 15 sets the MSS bit of the Status Byte of the master node. The commands to configure the status model for this example are provided in Status configuration (enable) commands (on page 642). When a current limit condition occurs in the slot 2 PSU module of node 15, the following sequence of events occurs:

- Node 15: Bit B1 or B2 of the slot 2 Measurement Event Current Limit Summary Register sets when the current limit event occurs.
- Node 15: Bit B1 (ILMT) of the slot 2 Measurement Event Register sets.
- Node 15: Bit B2 (SLOT2) of the Slot Summary Register sets.
- Node 15: Bit B9 (SLOT) of the Operation Status Register sets.
- Node 15: Bit B7 (OSB) of the Status Byte sets.
- System Summary Registers: Bit B1 (NODE15) of the System Summary Register 2 sets
- System Summary Registers: Bit B0 (EXT) of the System Summary Register sets.
- Master Node: Bit B1 (SSB) of the Status Byte Sets.
- Master Node: Bit B0 (MSS) of the Status Byte sets.

Node 15 enables:

```
node[15].slot[2].status.measurement.current_limit.enable = node[15].slot[2].status.measurement.curr
ent_limit.PSU1 + node[15].slot[2].status.measurement.current_limit.PSU2
node[15].slot[2].status.measurement.enable = node[15].slot[2].status.measurement.ILMT
node[15].slot[2].status.enable = node[15].slot[2].status.MSB
node[15].status.operation.slot.summary.enable = node[15].status.operation.slot.summary.SLOT2
node[15].status.operation.enable = node[15].status.operation.SLOT
node[15].status.node_enable = node[15].status.OSB
```

System Summary enables:

```
status.system[2].enable = status.system[2].NODE15
status.system[1].enable = status.system[1].EXT
```

Request Enable:

```
status.request_enable = status.SSB
```

Status configuration (enable) commands

For the following registers, the commands listed, which are sent from the master node, enable the appropriate register bits for the status model configuration example.

Node 15 status registers: The following commands enable the current limit events for PSU1 and PSU2 of node 15:

```
node[15].slot[2].status.measurement.current_limit.enable = 6
node[15].slot[2].status.measurement.enable = 2
node[15].slot[2].status.enable = 1
```

The affected status registers for the above commands are indicated in the following figure.

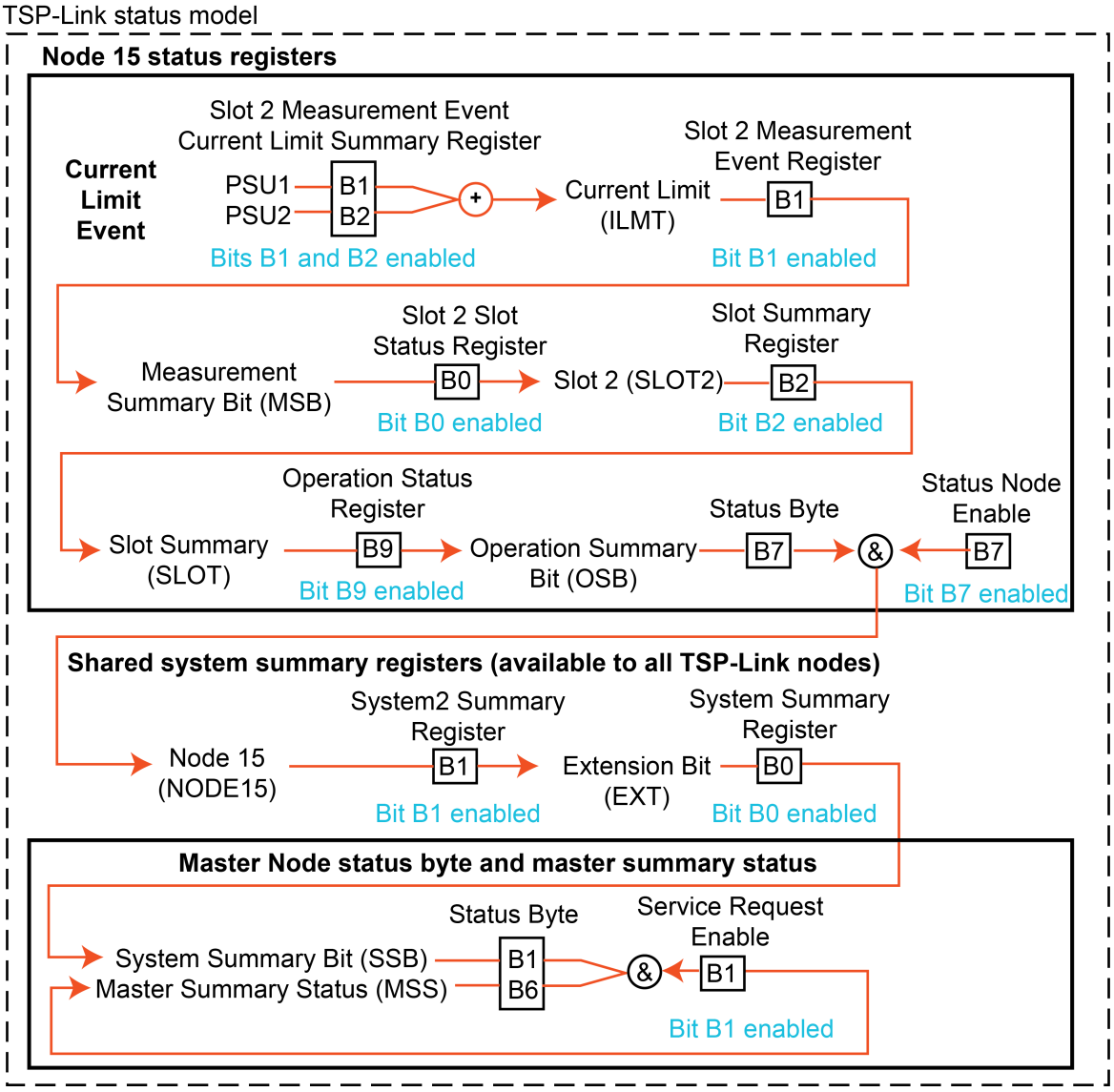
Master node system summary registers: The following commands enable the required system summary bits for node 15:

```
status.system2.enable = 2
status.system.enable = 1
```

Master node service request: The following command enables the service request for the measurement event:

```
status.request_enable = 1
```

The affected status registers for the commands are shown in the following figure.



System Summary registers

The TSP-Link™ expansion interface allows instruments to communicate with each other. The test system can be expanded to include up to 32 TSP-enabled instruments. In a TSP-Link system, one node (instrument) is the master and the other nodes are the subordinates. The master can control the other nodes (subordinates) in the system. See TSP-Link system expansion interface for details about the TSP-Link system.

There are five register sets associated with system status events. These registers summarize the system status for nodes that are connected to the TSP-Link interface. When a TSP-Link system is initialized, all nodes on the TSP-Link network share a copy of the System Summary registers. This feature allows all nodes to access the status models of other nodes.

In a TSP-Link system, you can configure the status model so that a status event in any node in the system can set the RQS (request for service) bit of the master node Status Byte. After detecting the service request, your program can then branch to an appropriate subroutine that services the request. See Status byte and service request (SRQ) (on page 638) for details.

For example, either of the following commands sets the EXT enable bit:

```
status.system.enable = status.system.EXT
status.system.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, refer to Read the registers (on page 637). For example, the following command reads the System Enable Register:

```
print(status.system.enable)
```

The used bits of the System Summary registers are:

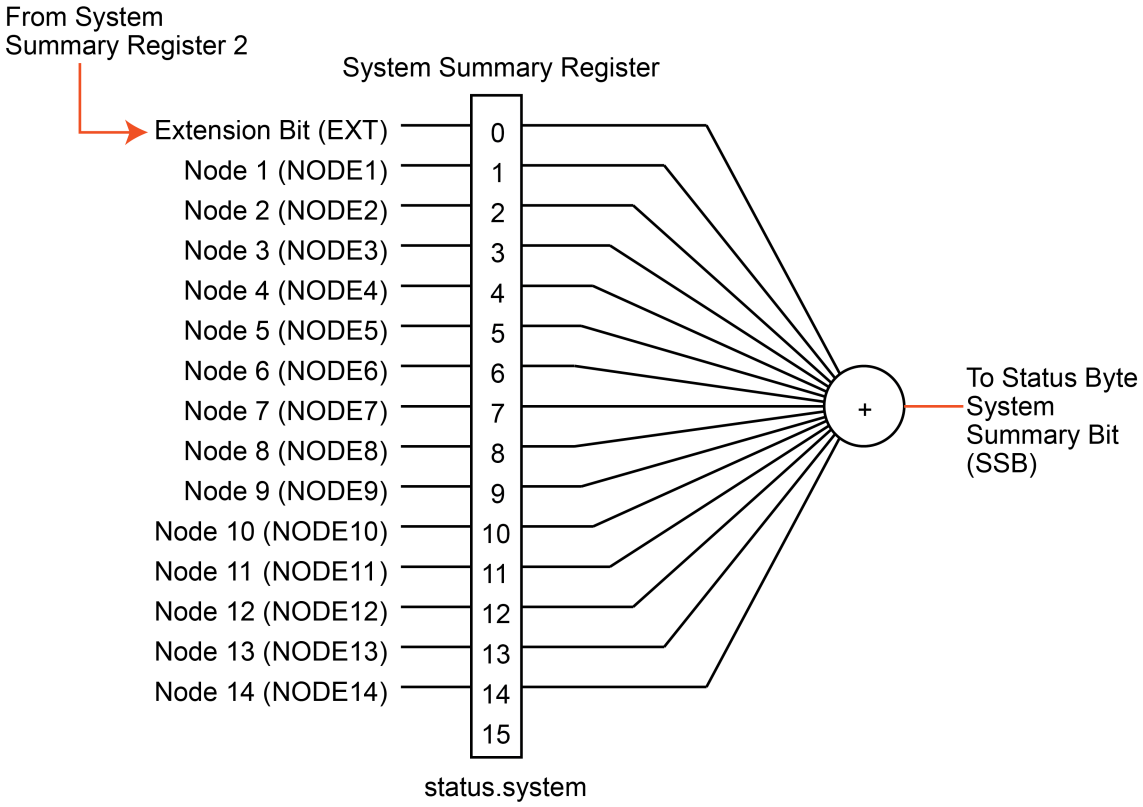
- **Bit B0, Extension Bit (EXT):** Set bit indicates that an extension bit from another System Status register is set.
- **Bits B1 to B14 NODE*n*:** Indicates a bit on TSP-Link node *n* has been set (*n* = 1 to 64) (note that `status.system5` does not use bits B9 through B15).

The System Summary registers are represented in the following diagrams.

For more information on TSP-Link, refer to TSP-Link system expansion interface.

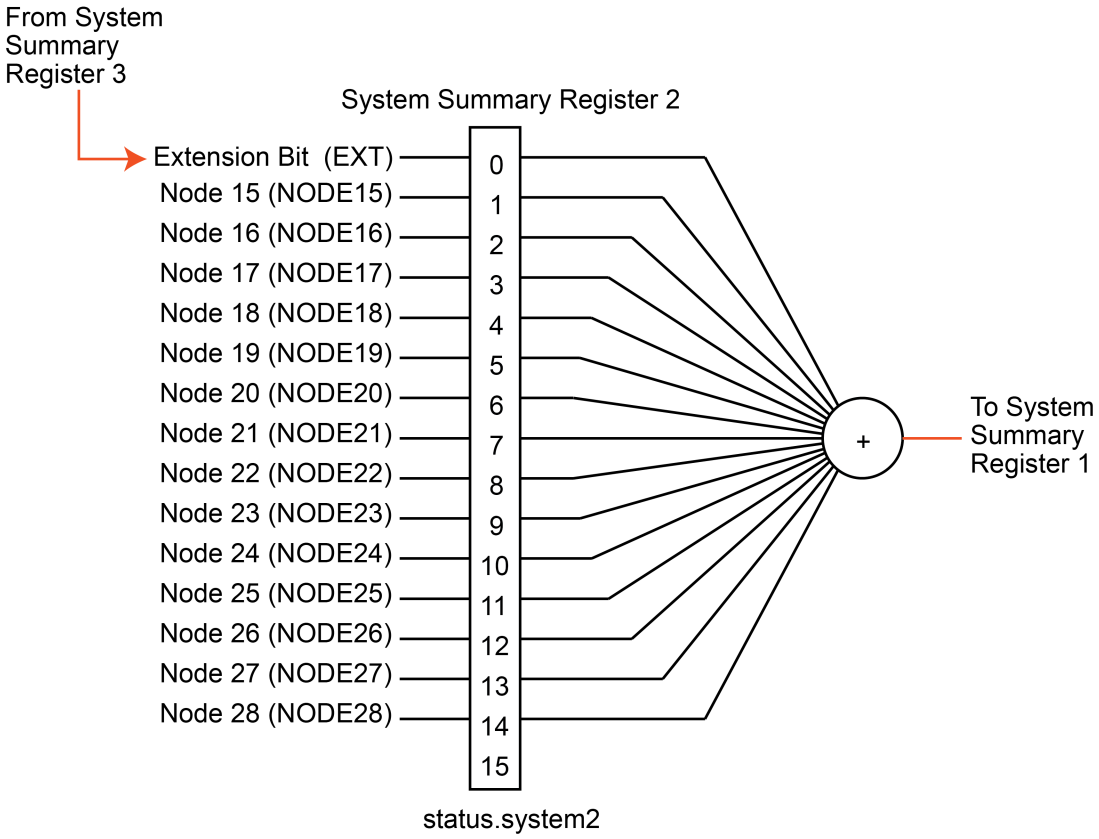
System summary 1

This register is available on all TSP-Link nodes.



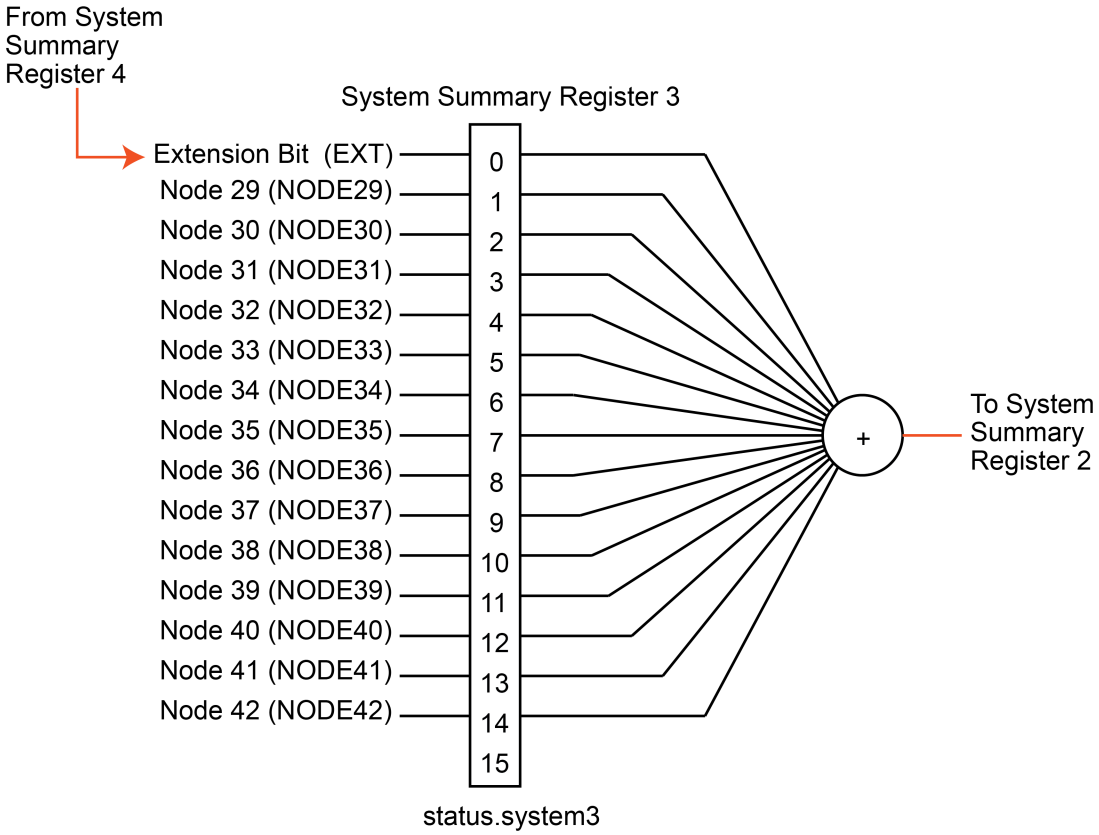
System summary 2

This register is available on all TSP-Link nodes.



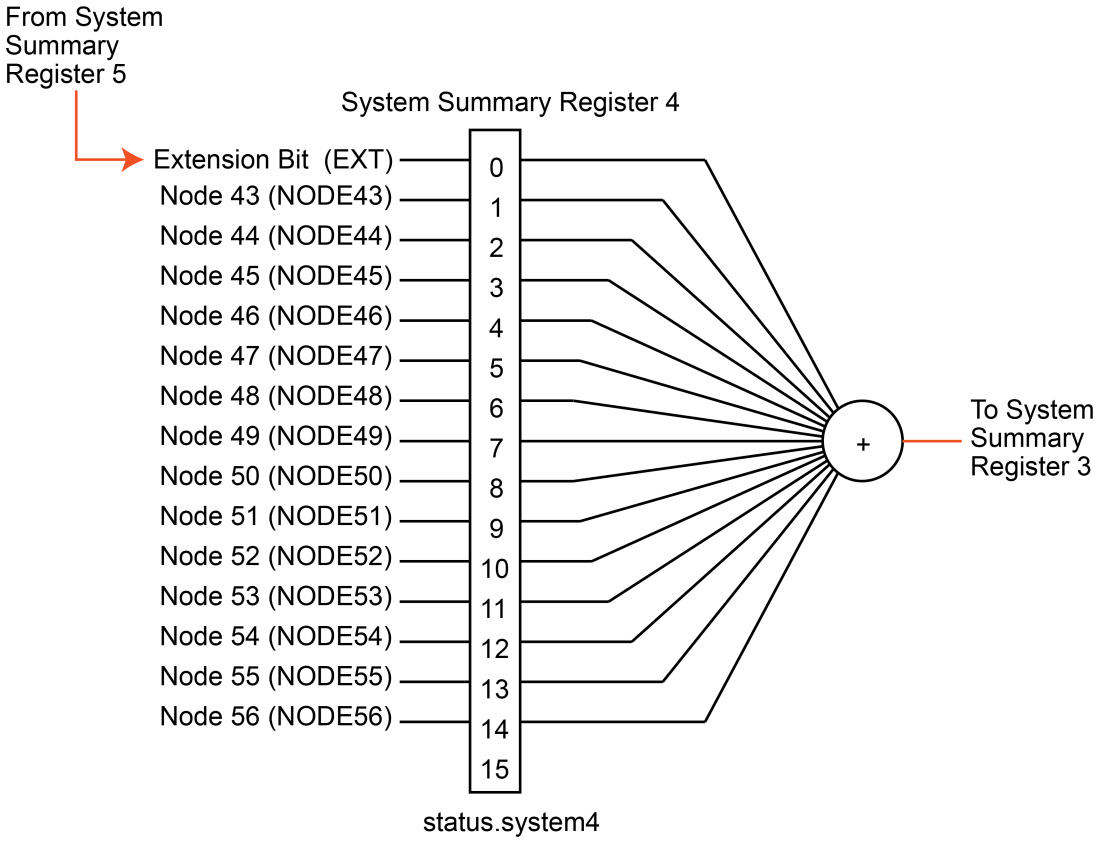
System summary 3

This register is available on all TSP-Link nodes.



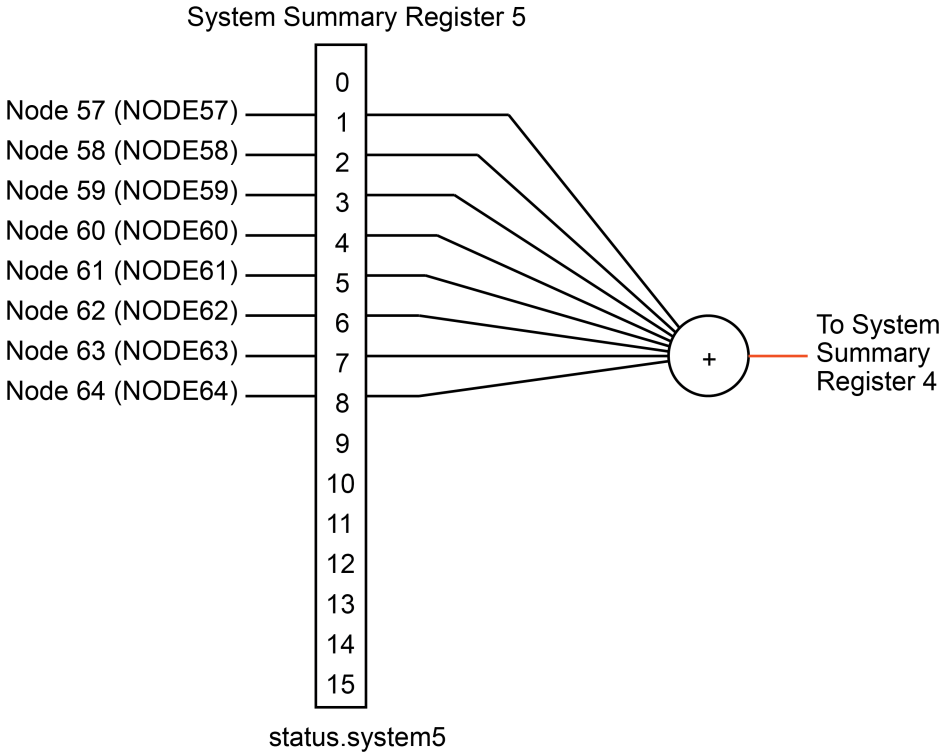
System summary 4

This register is available on all TSP-Link nodes.

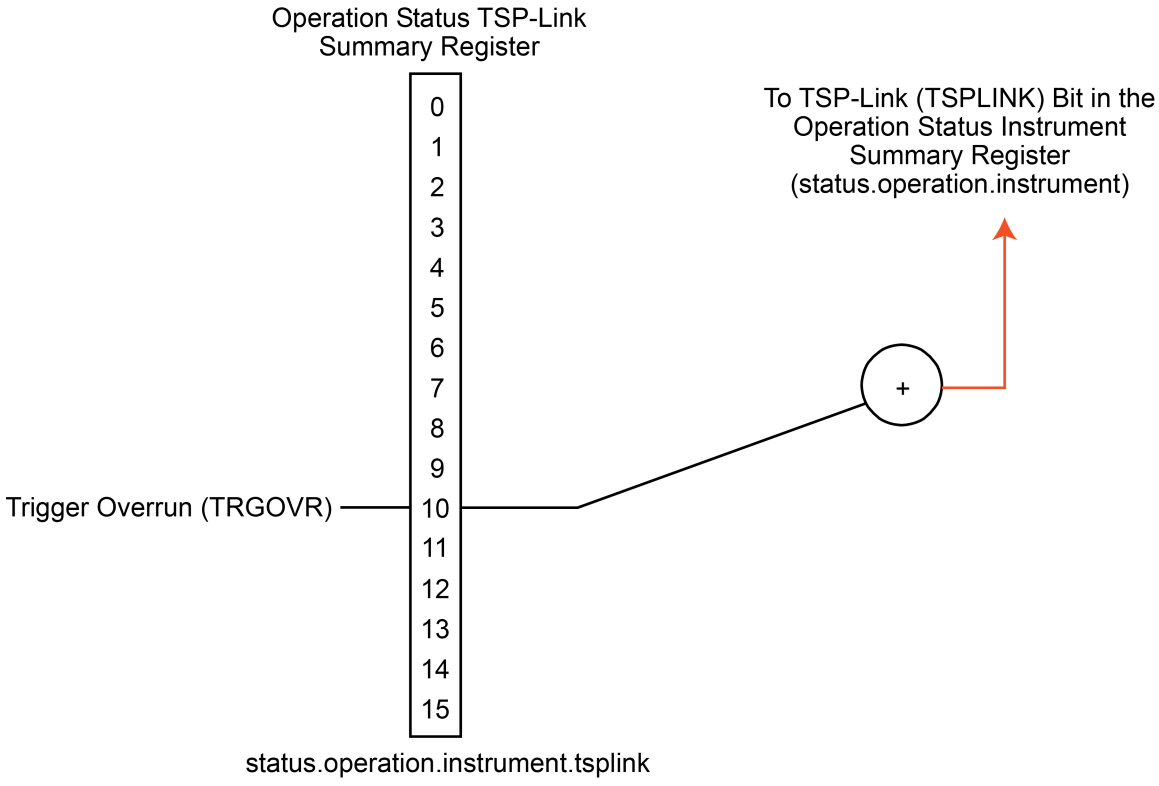


System summary 5

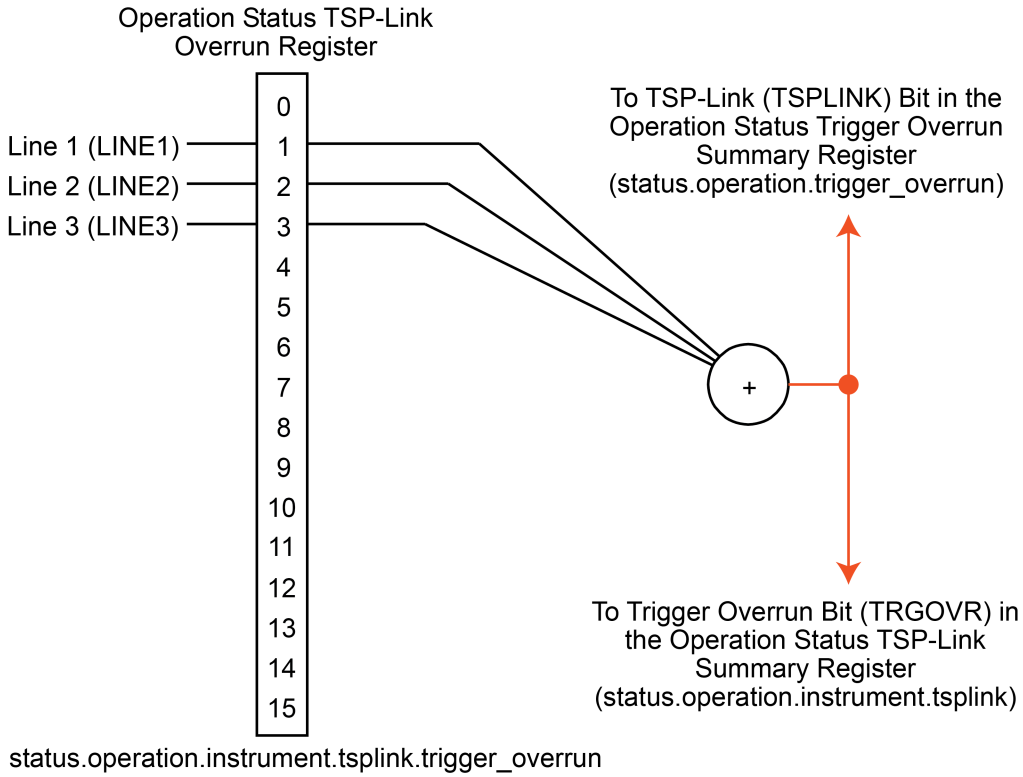
This register is available on all TSP-Link nodes.



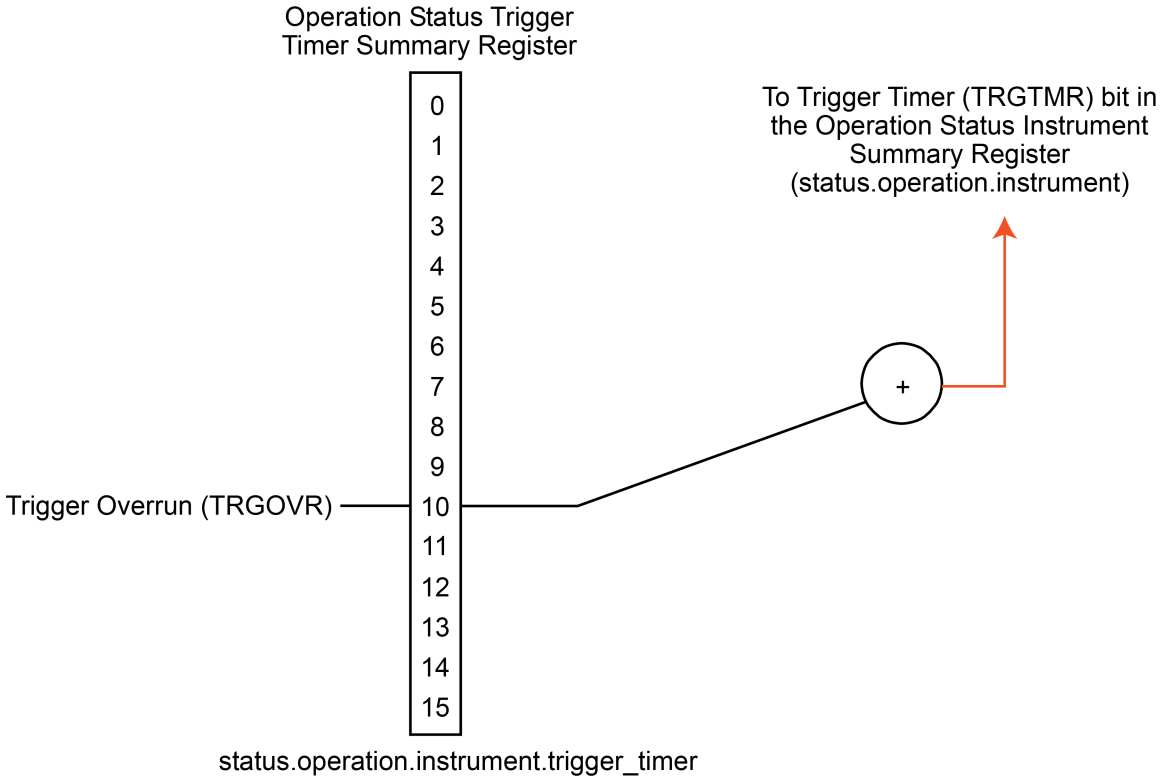
Operation Status TSP-Link Summary register



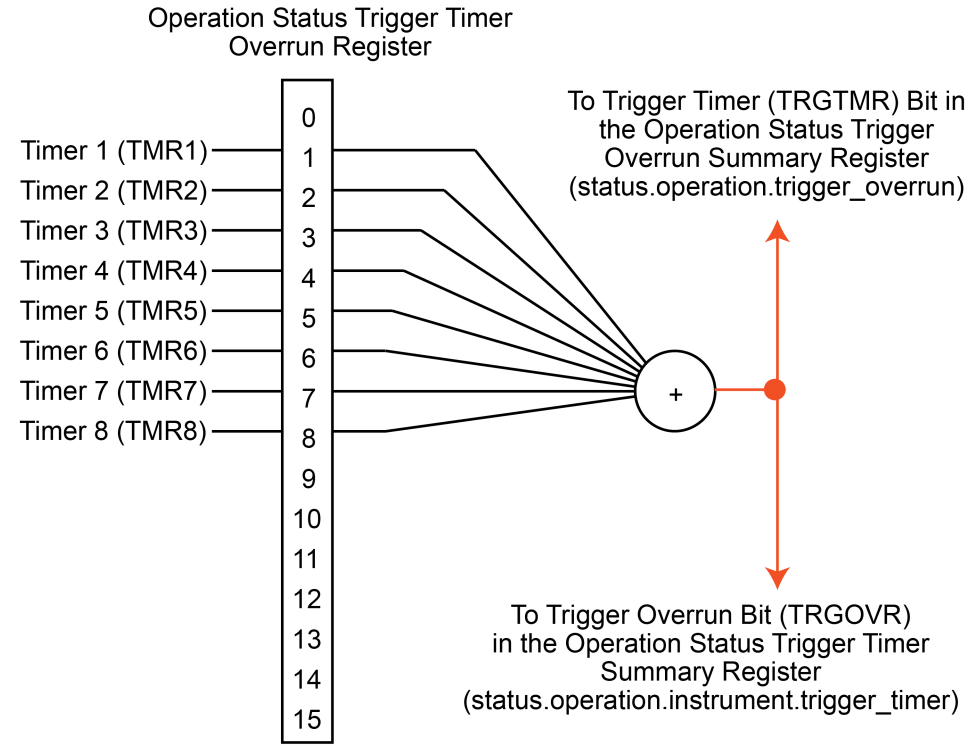
Operation Status TSP-Link Overrun Register



Operation Status Trigger Timer Summary Register

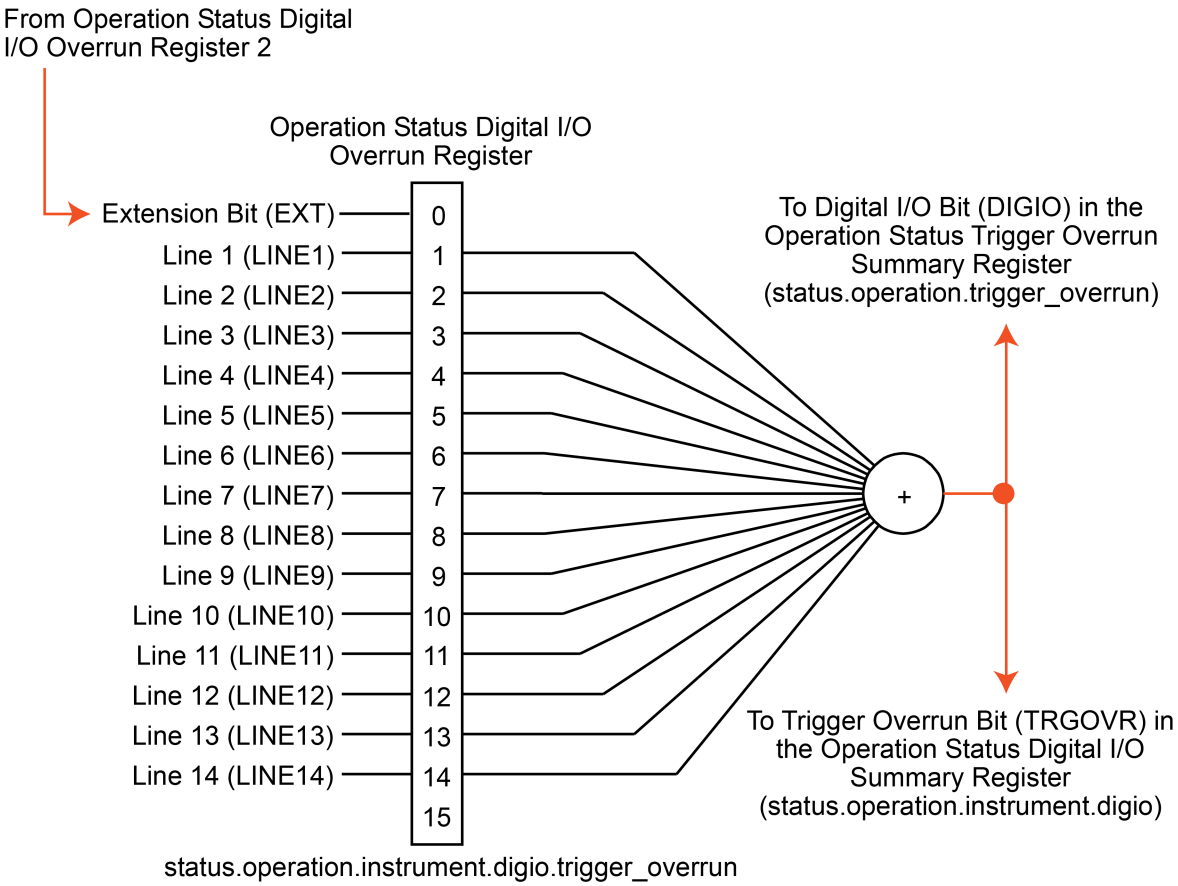


Operation Status Trigger Timer Overrun Register

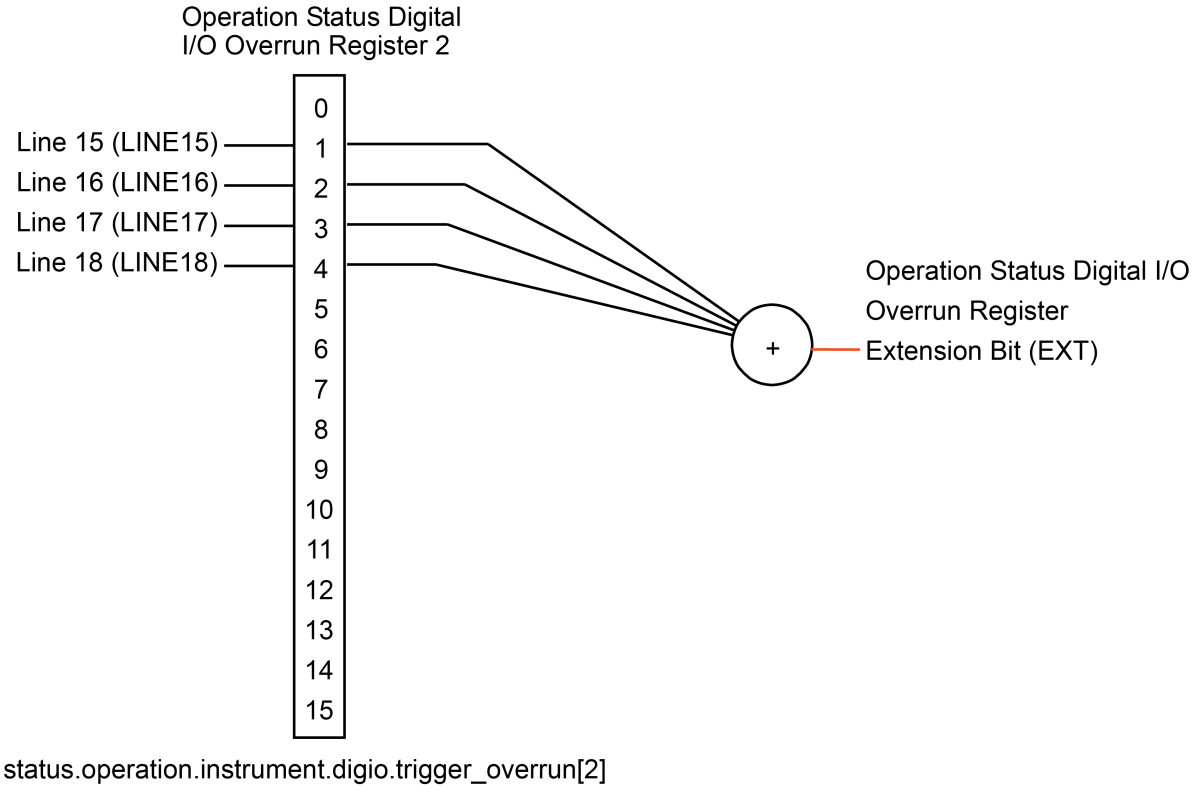


status.operation.instrument.trigger_timer.trigger_overrun

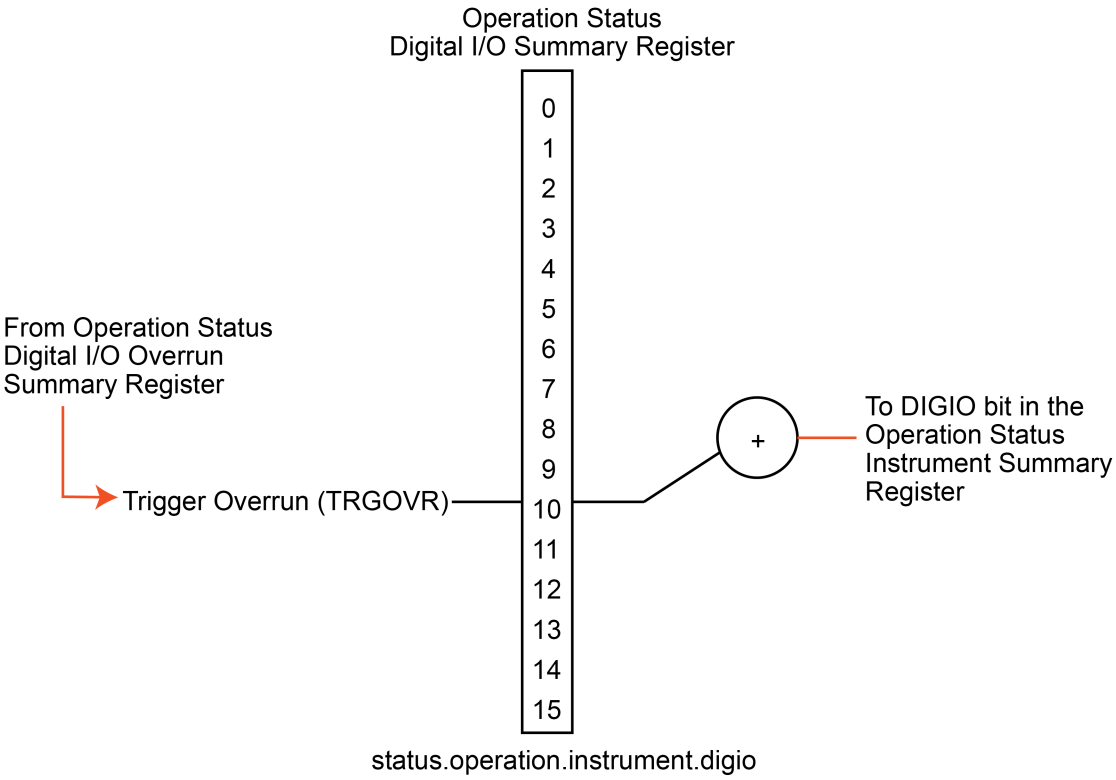
Operation Status Digital I/O Overrun Register



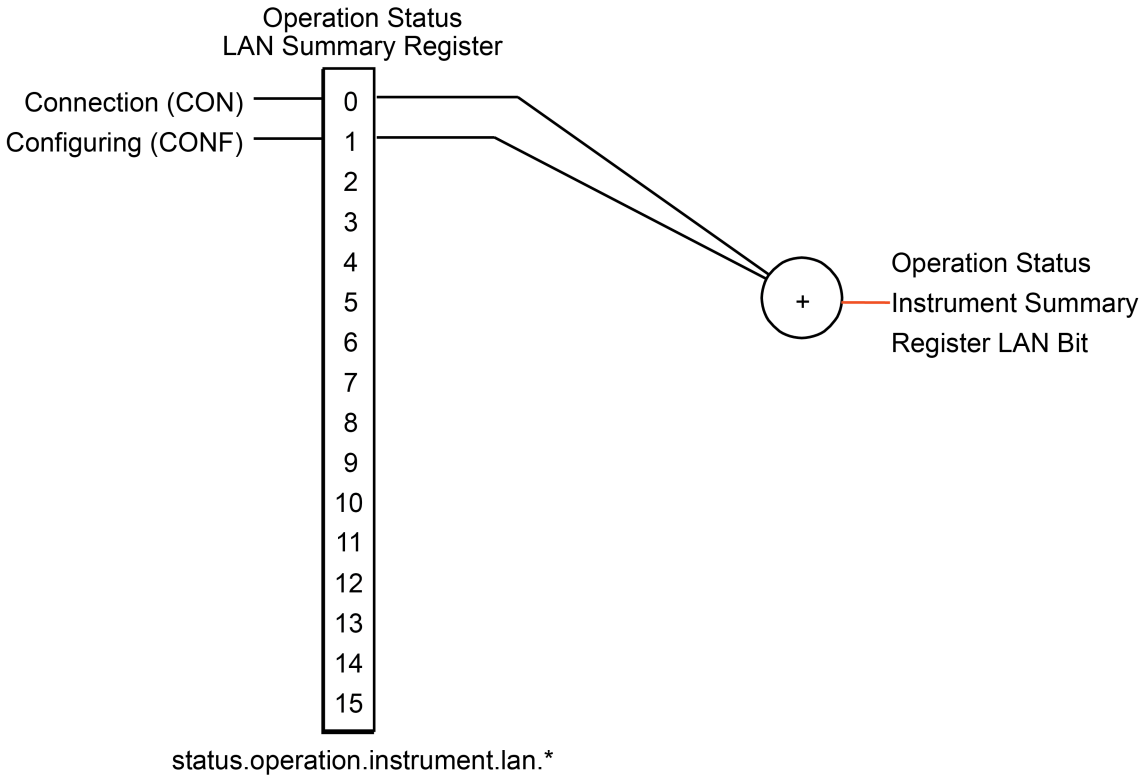
Operation Status Digital I/O Overrun Register 2



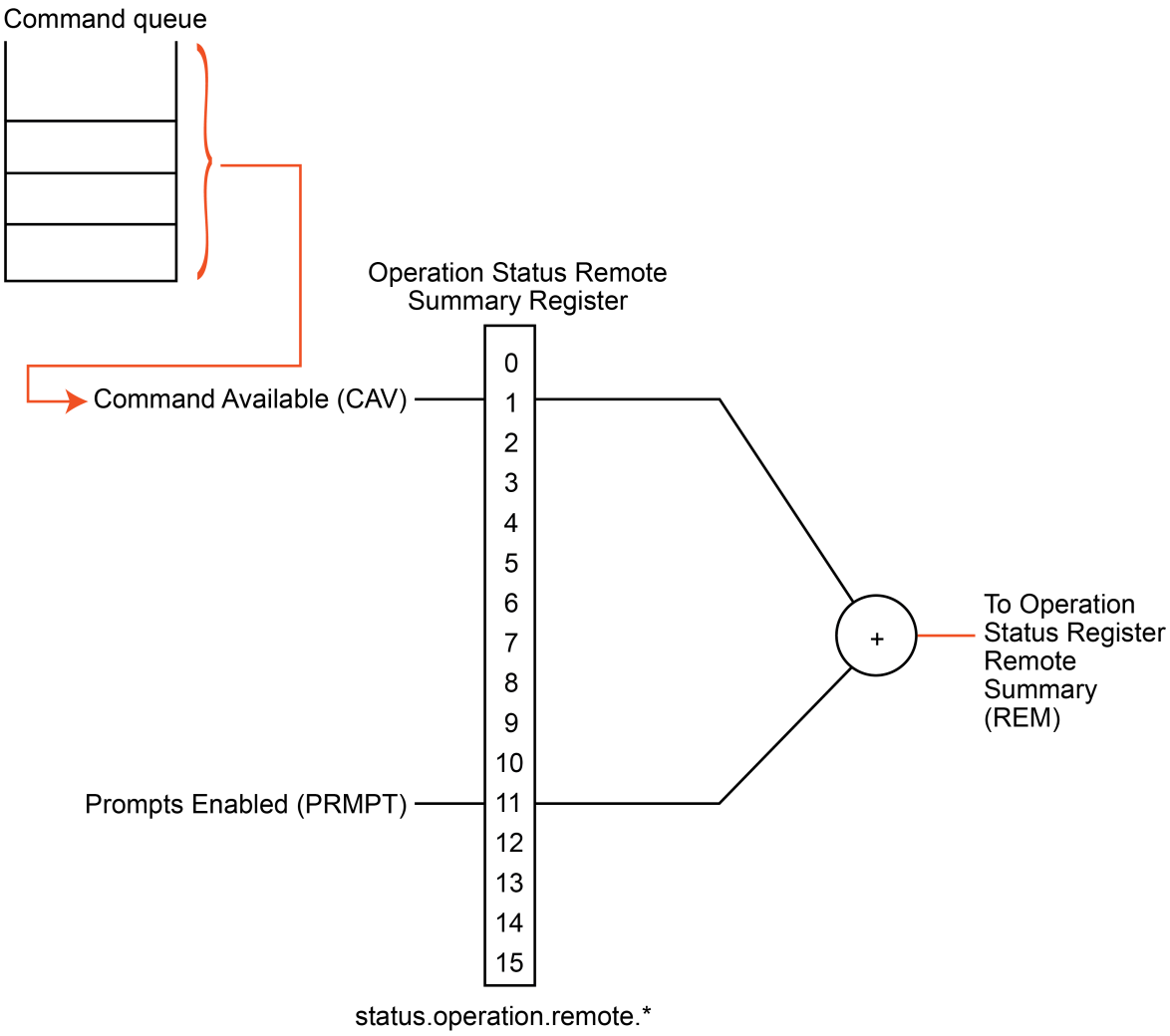
Operation Status Digital I/O Summary Register



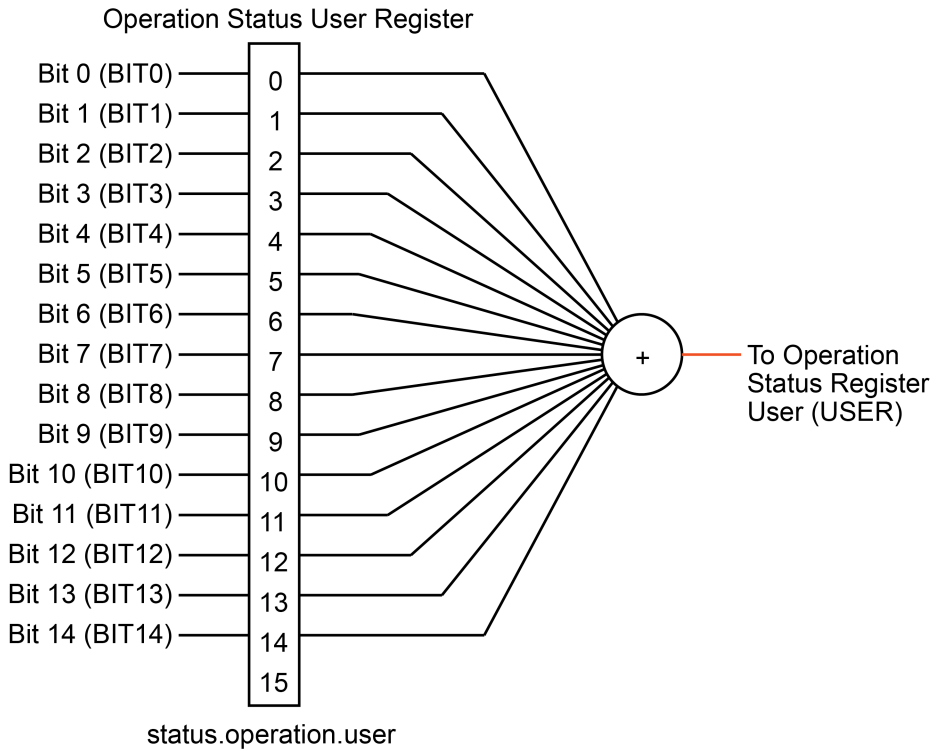
Operation Status LAN Summary Register



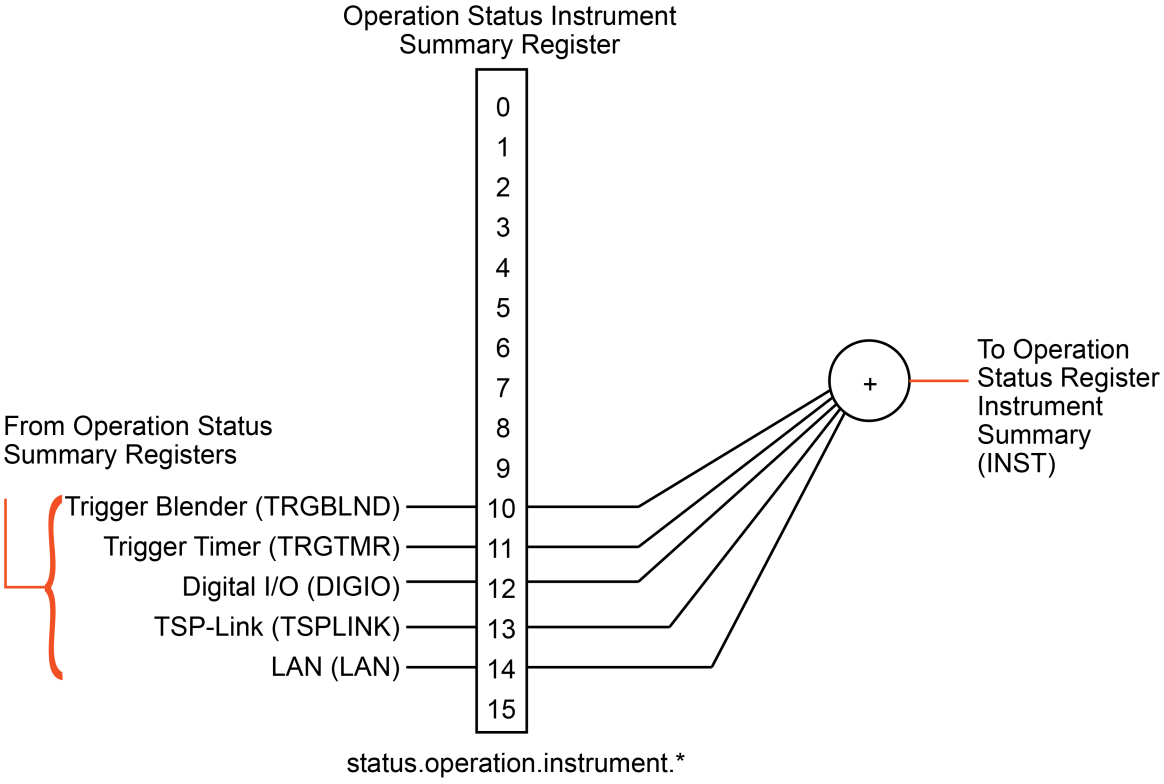
Operation Status Remote Summary Register



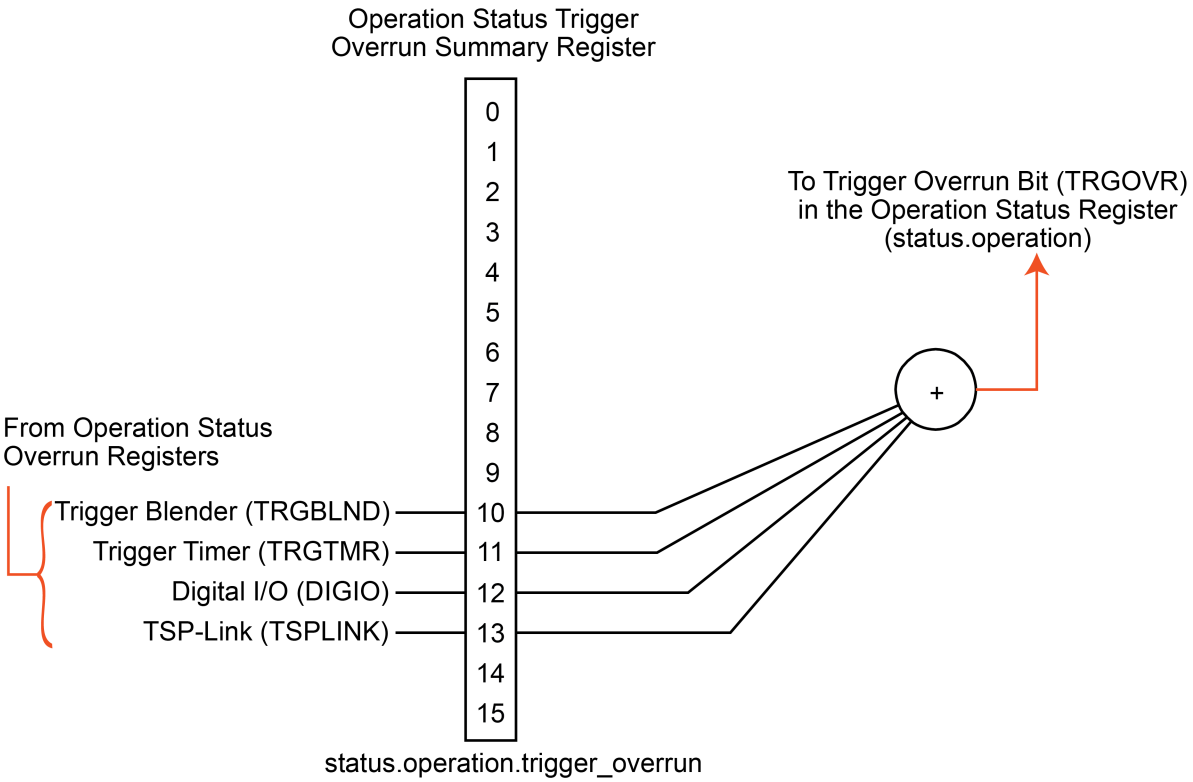
Operation Status User Register



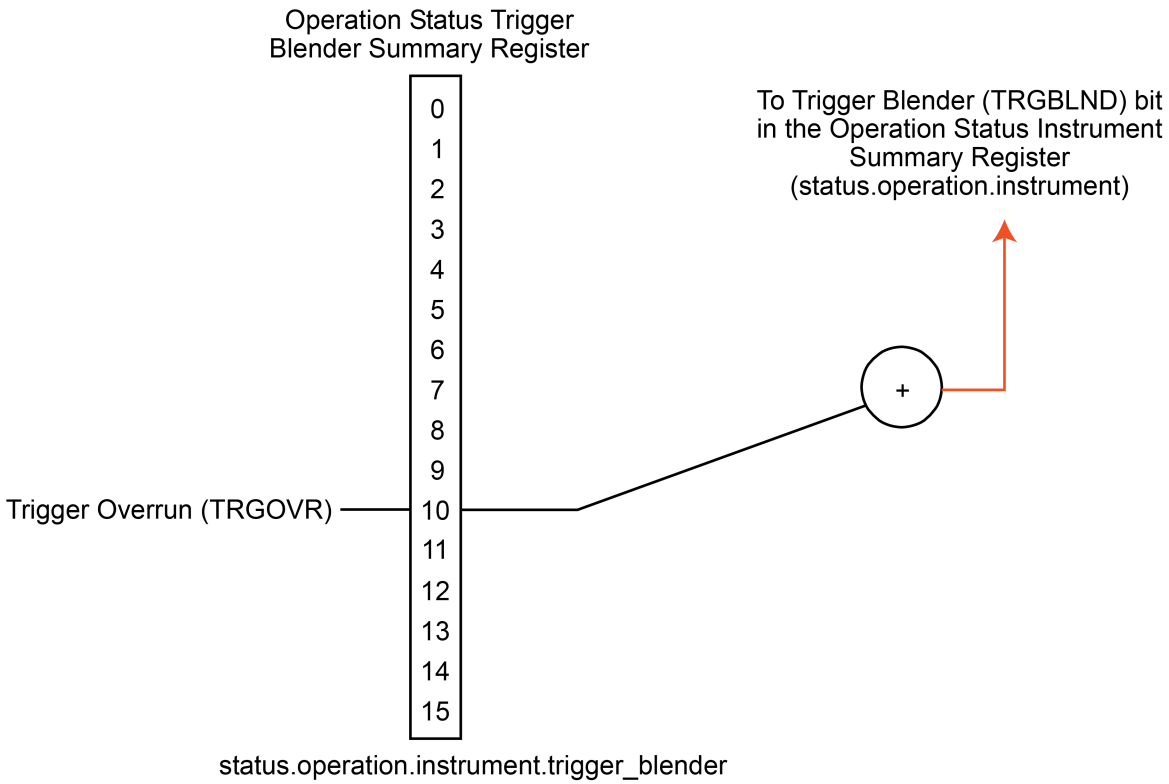
Operation Status Instrument Summary Register



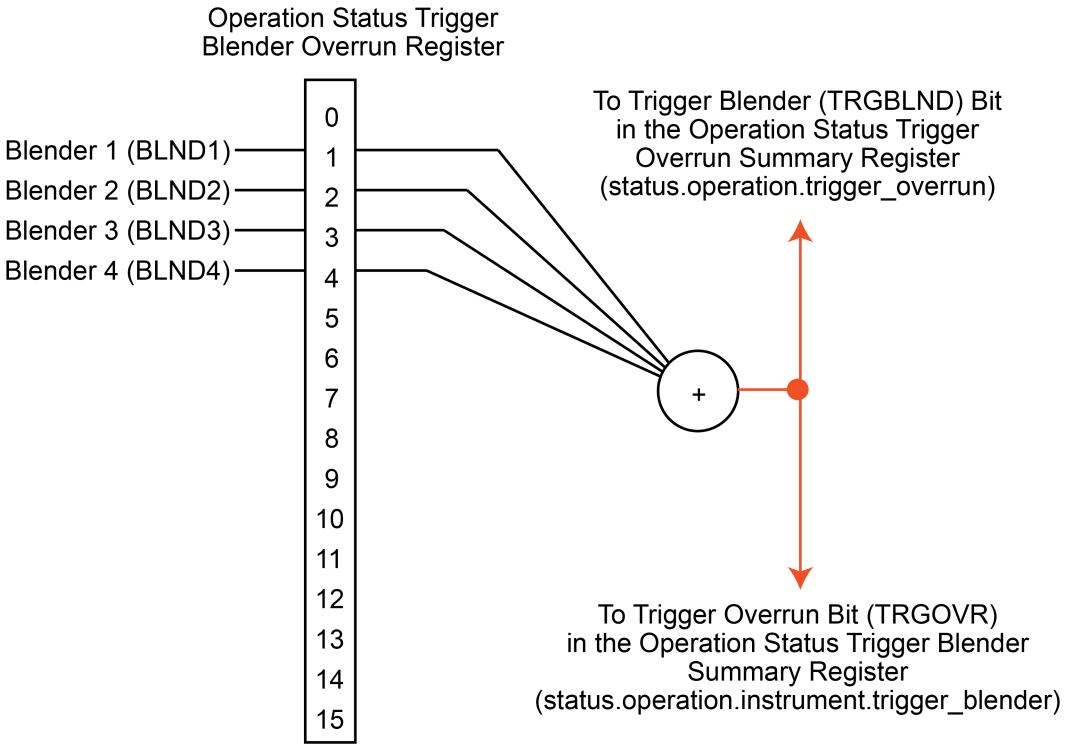
Operation Status Trigger Overrun Summary Register



Operation Status Trigger Blender Summary Register



Operation Status Trigger Blender Overrun Summary Register

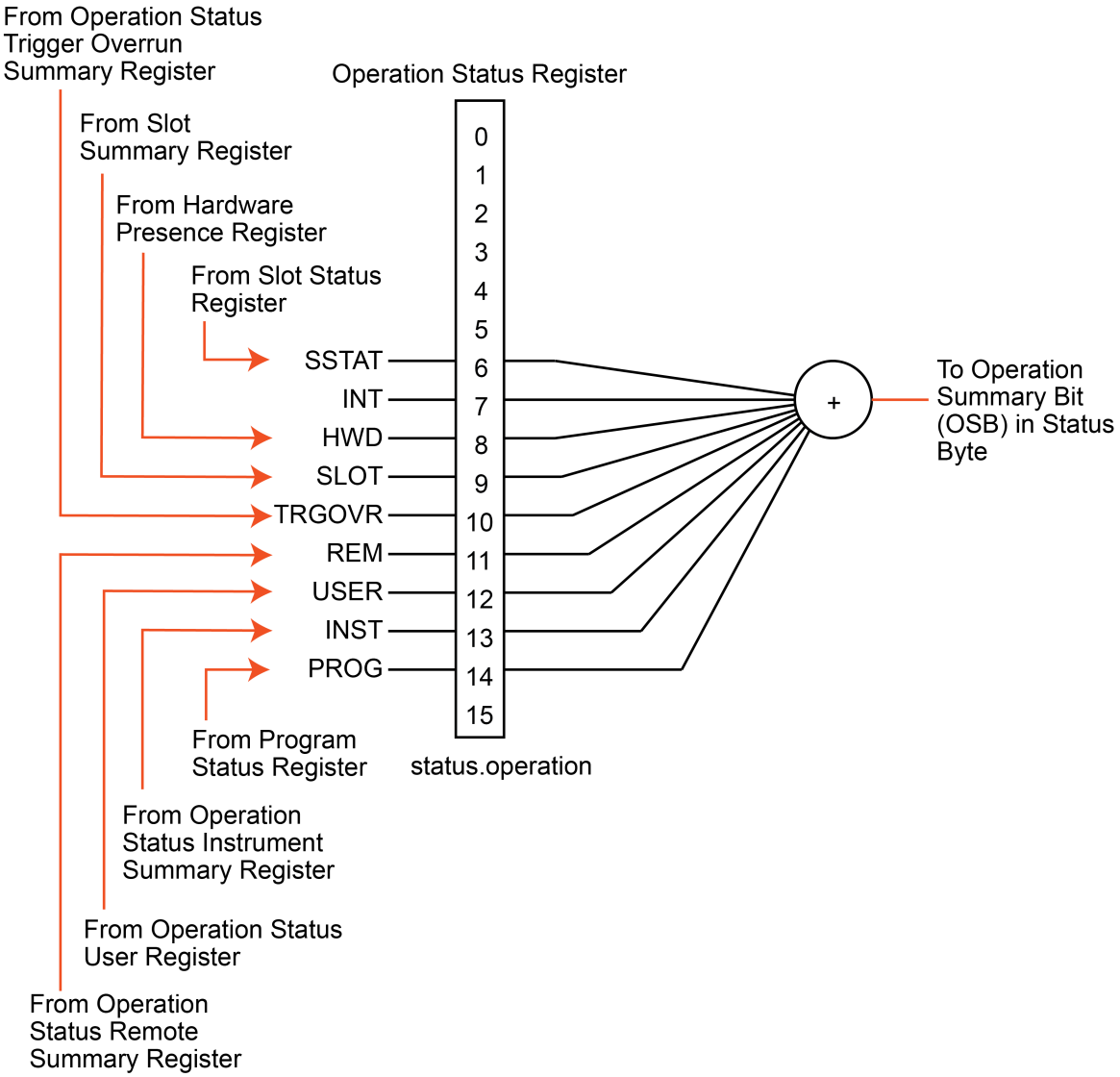


status.operation.instrument.trigger_blender.trigger_overrun

Operation Status Registers

You can program the bits in the Operation Status Registers to be cleared or set when an event occurs.

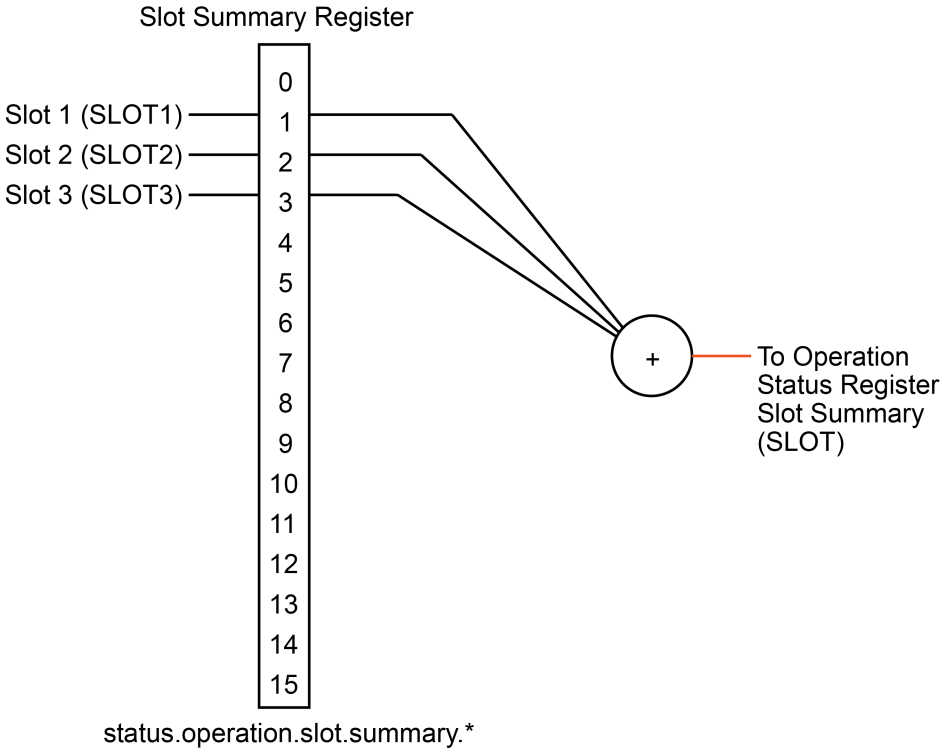
When an enabled Operation Status Register bit is set (because the enabled event occurs), the corresponding bit B7 (OSB) of the Summary Bit in the Status Byte is set. The corresponding Operation Event Register Condition Register reflects the present status of the instrument, so it will be set while the event occurs.



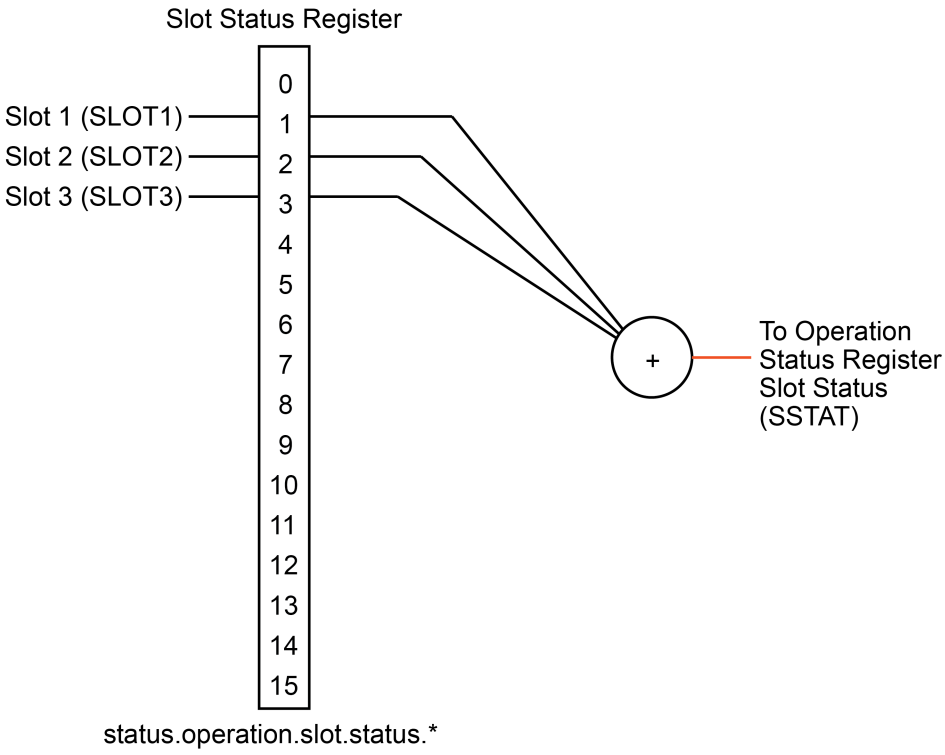
Bit name	Description
SSTAT	Slot Status
INT	Interlock
HWD	Hardware Detect
SLOT	Slot Summary
TRGOVR	Trigger Overrun

Bit name	Description
REM	Remote Summary
USER	User
INST	Instrument Summary
PROG	Program Status

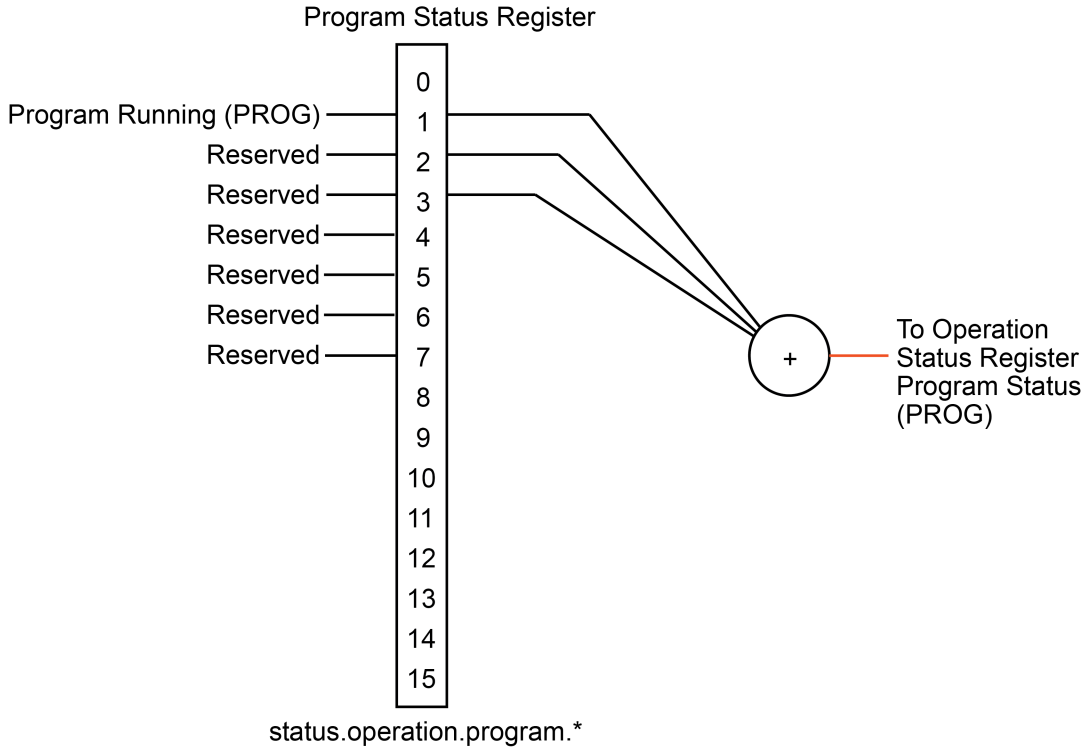
Slot Summary Register



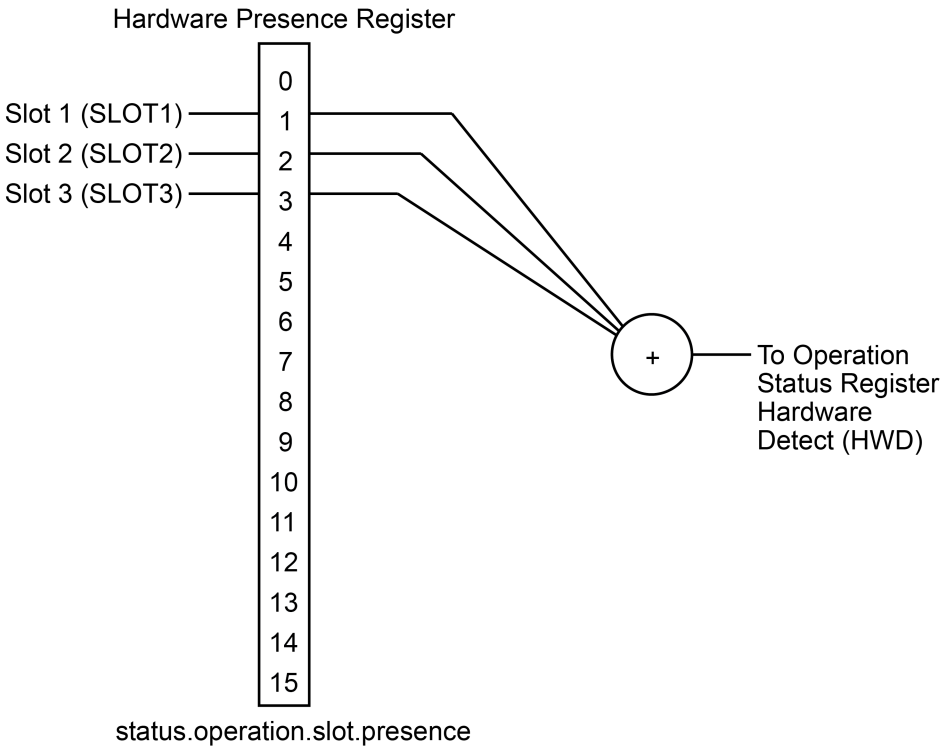
Slot Status Register



Program Status Register



Hardware Presence Register



MSMU60-2 and MSMU200-2 status model

Measurement Event Registers

The following register sets associated with measurement event status:

- Measurement Event Current Limit Summary register
- Measurement Event Voltage Limit Summary register
- Measurement Event Reading Overflow Summary register
- Measurement Event SMU 1 Summary register
- Measurement Event Instrument Summary register

These registers are sent to the Measurement Event register, which feeds to bit B0 (MSB) of the Slot Status Register. The bits used in the Measurement Event registers are:

- **Bit B0, Voltage Limit (VLMT):** Set bit indicates that the voltage limit was exceeded. This bit is updated only when either a measurement is made or the `slot[Z].smu[X].source.atlimit` attribute is read.
- **Bit B1, Current Limit (ILMT):** Set bit indicates that the current limit was exceeded. This bit is updated only when either a measurement is made or the `slot[Z].smu[X].source.atlimit` attribute is read.
- **Bit B7, Reading Overflow (ROF):** Set bit indicates that an overflow reading has been detected.
- **B11, Interlock (INT):** Set bit indicates that the interlock has been asserted.
- **B12, Instrument Summary (INST):** Set bit indicates that a bit in the measurement instrument summary register is set.

Bits can be set by using enumerated values or numeric parameter values. For example, either of the following commands sets the `VOLTAGE_LIMIT` enable bit:

```
slot[1].status.measurement.enable = slot[1].status.measurement.VOLTAGE_LIMIT
slot[1].status.measurement.enable = 1
```

When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Read registers. For example, the following command reads the Measurement Event enable register:

```
print(slot[1].status.measurement.enable)
```

For additional detail, see Program enable and transition registers (on page 636).

For more information about the Measurement Event Registers, refer to Status register set contents (on page 634) and the figures in this section.

Measurement Event Summary registers

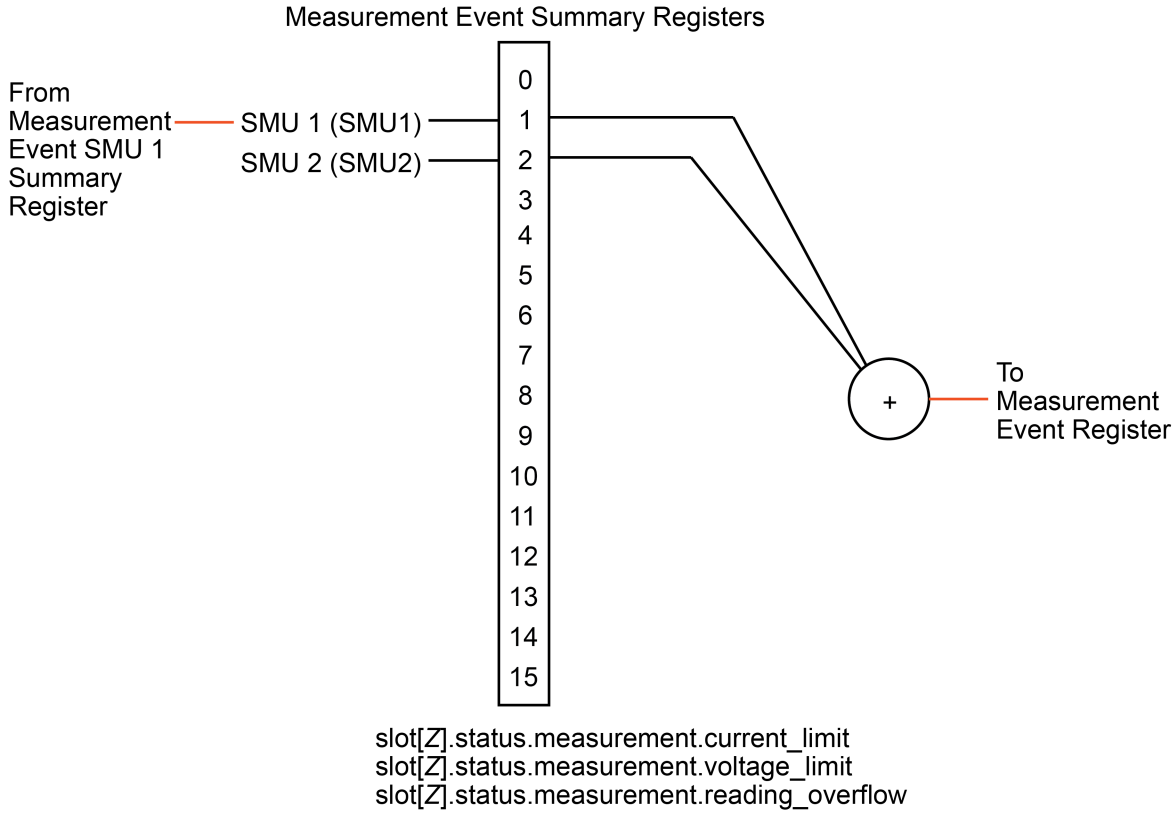
Measurement Event Summary registers are provided for current limit, voltage limit, reading overflow, and buffer available.

The input to the Measurement Event Summary Register comes from the appropriate register in the Measurement Event SMU1 Summary Register:

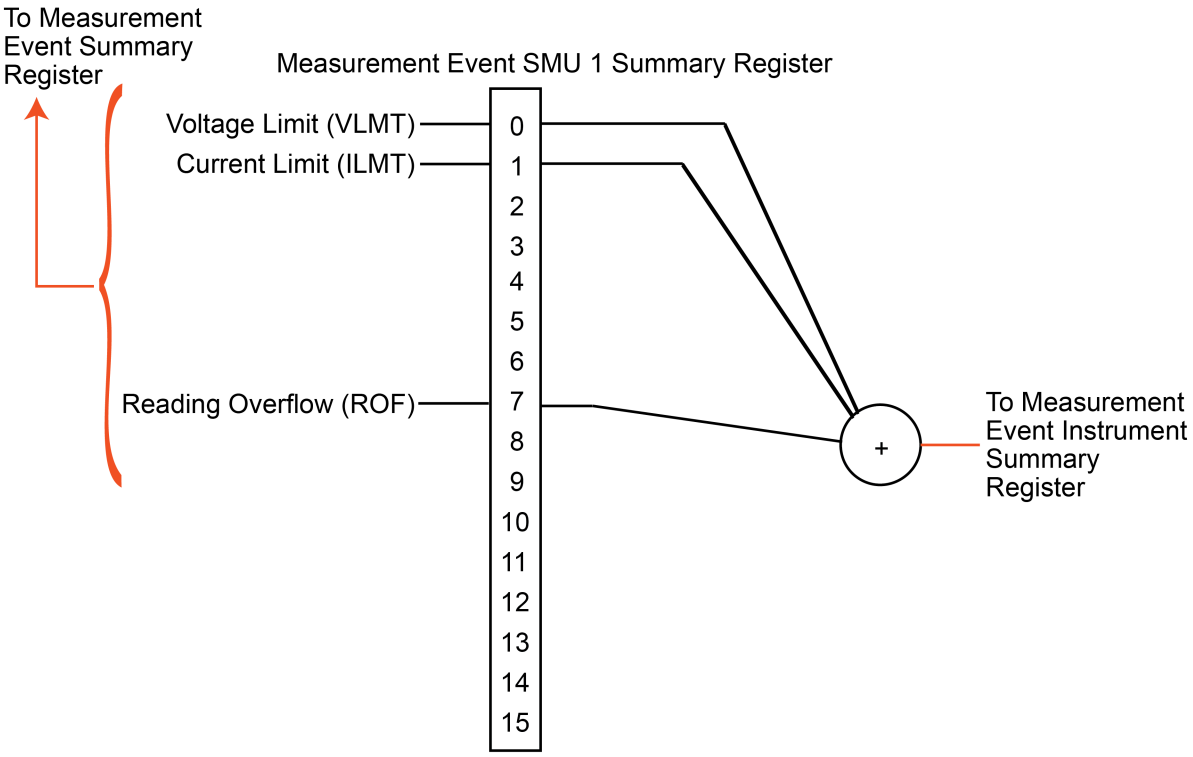
- **Voltage Limit (VLMT):** Bit 0
- **Current Limit (ILMT):** Bit 1
- **Reading Overflow (ROF):** Bit 7

The output of the Measurement Event Summary register goes to the appropriate register in the Measurement Event register:

- **Voltage Limit (VLMT):** Bit 0
- **Current Limit (ILMT):** Bit 1
- **Reading Overflow (ROF):** Bit 7

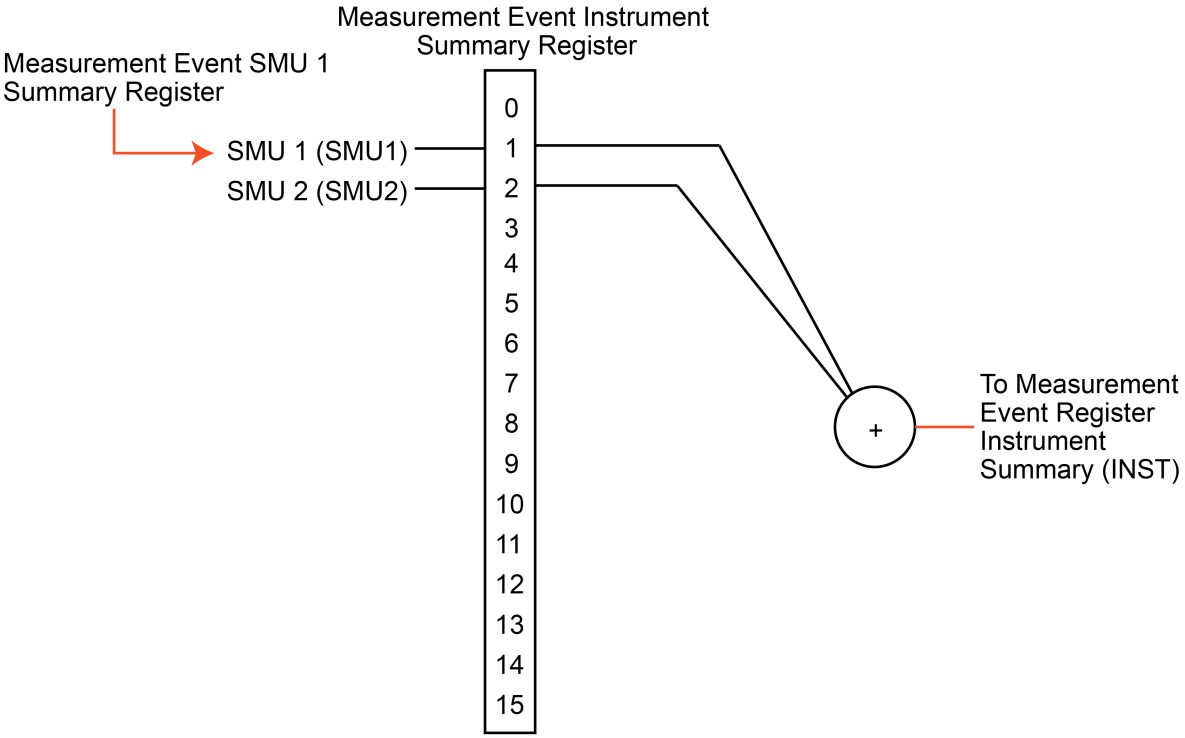


Measurement Event SMU 1 Event Summary Register



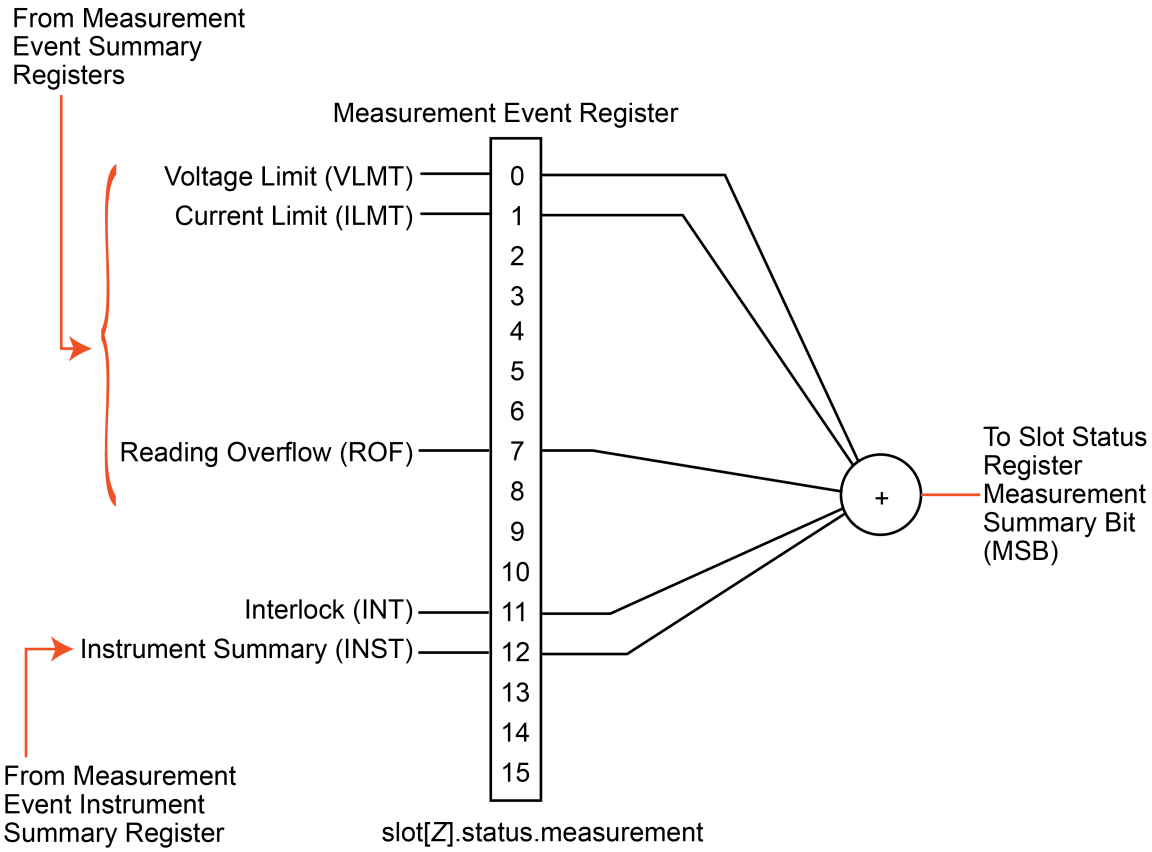
slot[*Z*].status.measurement.instrument.smu[*X*]

Measurement Event Instrument Summary Register



slot[*Z*].status.measurement.instrument

Measurement Event Register

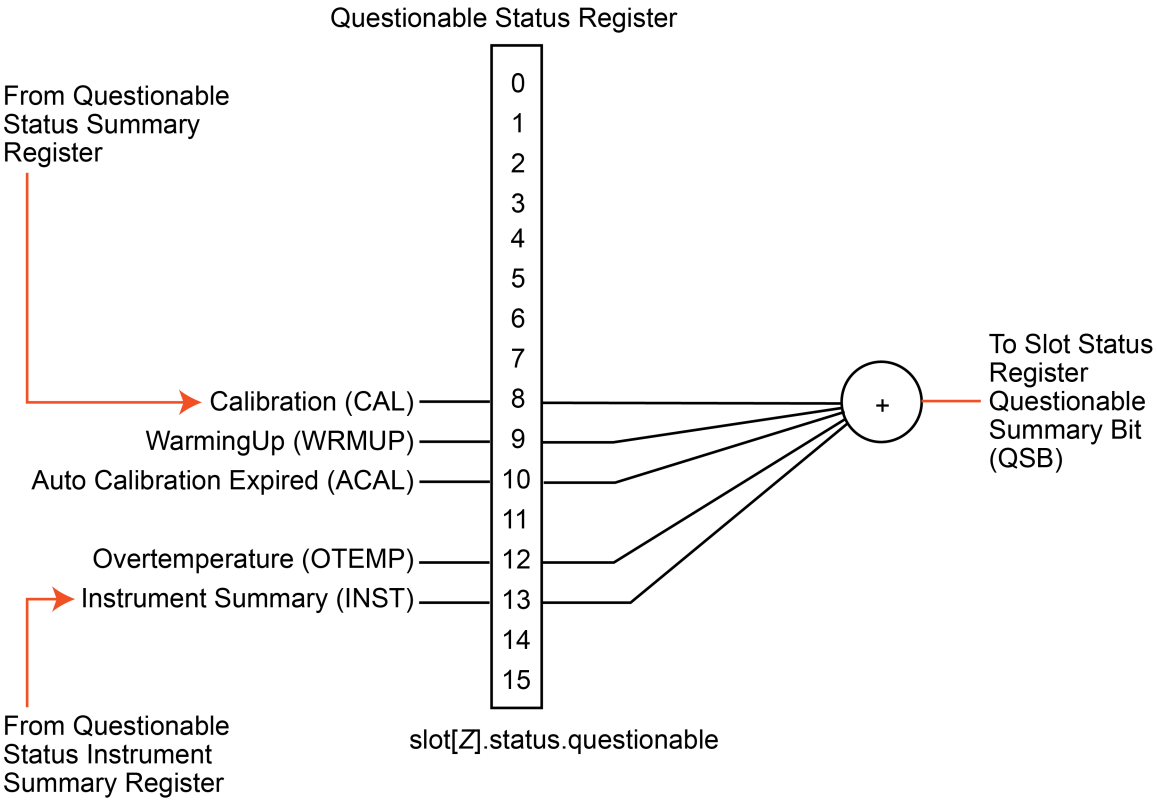


Questionable Status Register

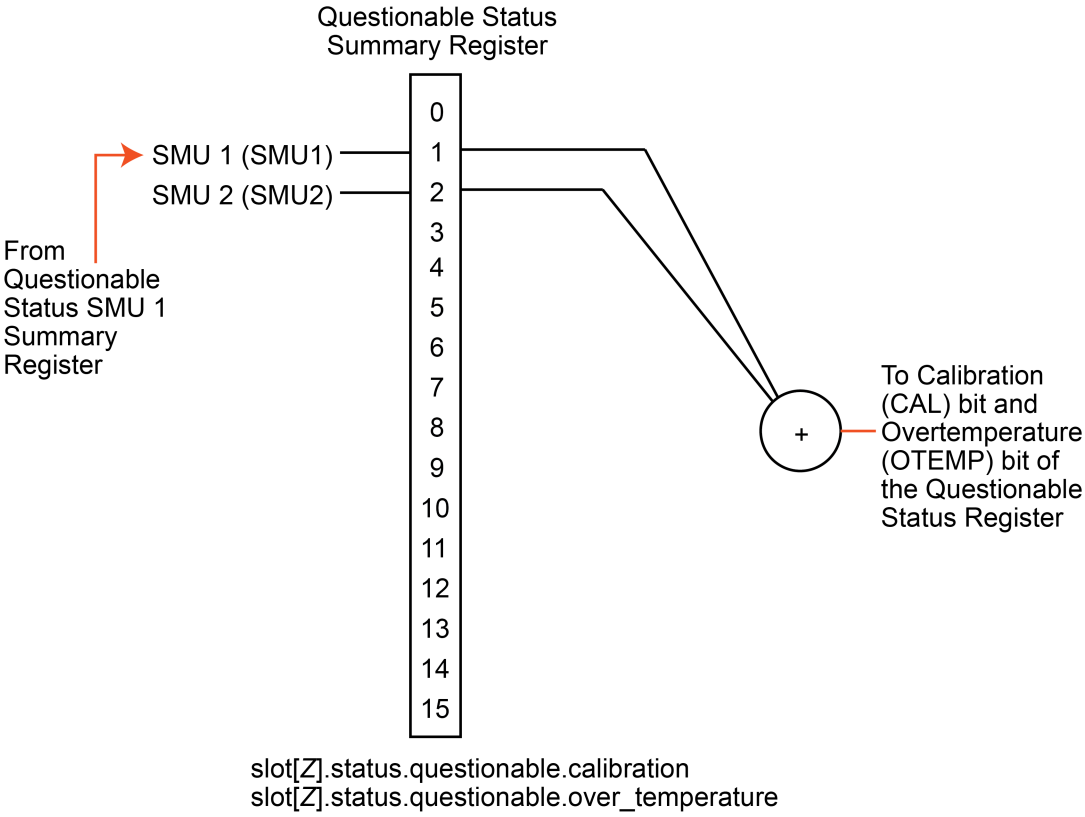
This register set feeds to bit B3 (QSB) of the Slot Status Register. The bits used in the Questionable Status Register set are described as follows:

- **Bit B8, Calibration (CAL):** Set bit indicates that calibration is questionable.
- **Bit B9, Warming Up (WRMUP):** Set bit indicates that the instrument is warming up.
- **Bit B10, Auto Calibration Expired (ACAL):** Set bit indicates that the automatic calibration constants on the instrument have expired.
- **Bit B12, Overtemperature (OTEMP):** Set bit indicates that an overtemperature condition was detected.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Questionable Status Instrument Summary Register is set.

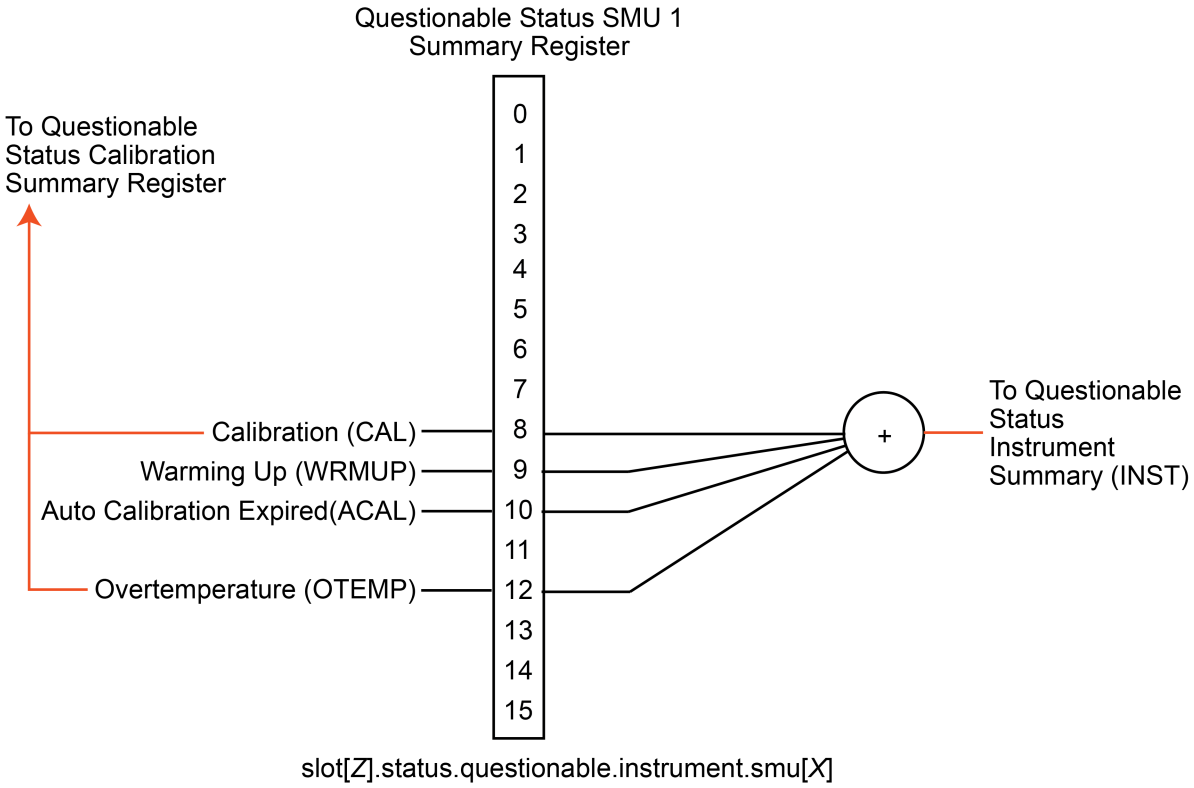
You can read or set the bits using the slot[Z].status.questionable.* (on page 561) command.



Questionable Status Summary Register

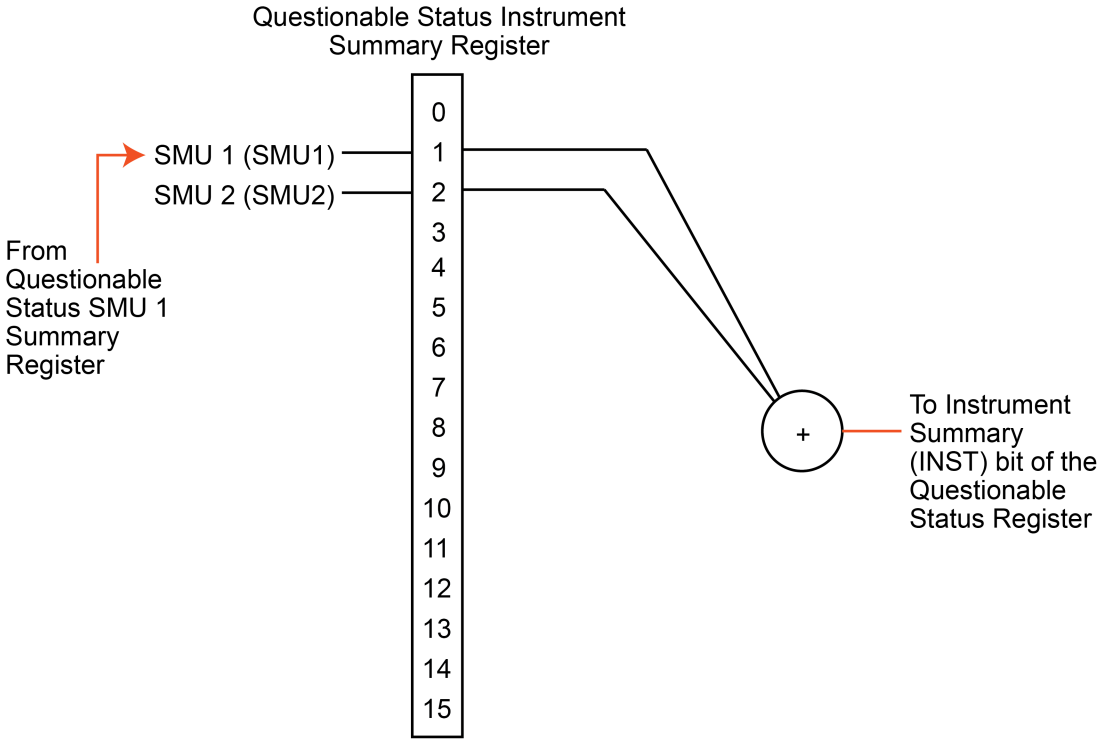


Questionable Status SMU 1 Summary Register



slot[*Z*].status.questionable.instrument.smu[*X*]

Questionable Status Instrument Summary Register



slot[*Z*].status.questionable.instrument

SMU Operation Status Registers

The following register sets associated with operation status:

- Operation Status Calibration Summary register
- Operation Status Sweeping Summary register
- Operation Status Measuring Summary register
- Operation Status Pulse Limits Exceeded Summary register
- Operation Status Instrument Summary register

These registers are sent to the Operation Status register, which feeds to bit B7 (OSB) of the Slot Status Register. The bits used in the Operation Status register are:

- **Bit B0, Calibrating (CAL):** Set bit indicates that one or more channels have calibration unlocked.
- **Bit B3, Sweeping (SWE):** Set bit indicates that one or more channels are sweeping.
- **Bit B4, Measuring (MEAS):** Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement.
- **Bit B7, Pulse Limits Exceeded (PLE):** Set bit indicates that SMU pulse limits have been exceeded.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that an enabled bit in the Operation Status Instrument Summary Register is set.

Bits can be set by using enumerated values or numeric parameter values. For example, either of the following commands sets the measuring enable bit:

```
slot[2].status.operation.enable = slot[2].status.operation.MEAS
slot[2].status.measurement.enable = 16
```

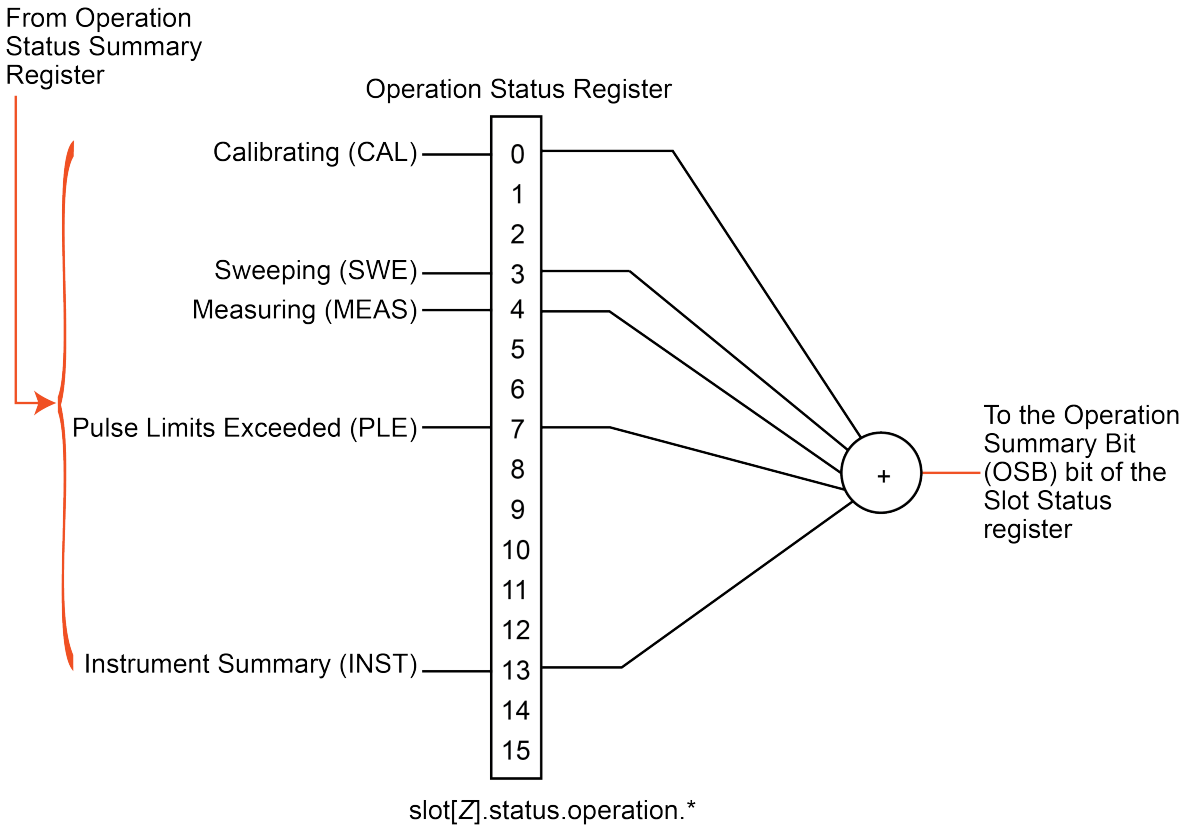
When reading a register, a numeric value is returned. The binary equivalent of this value indicates which bits in the register are set. For details, see Read registers. For example, the following command reads the Operation Status enable register:

```
print(slot[2].status.operation.enable)
```

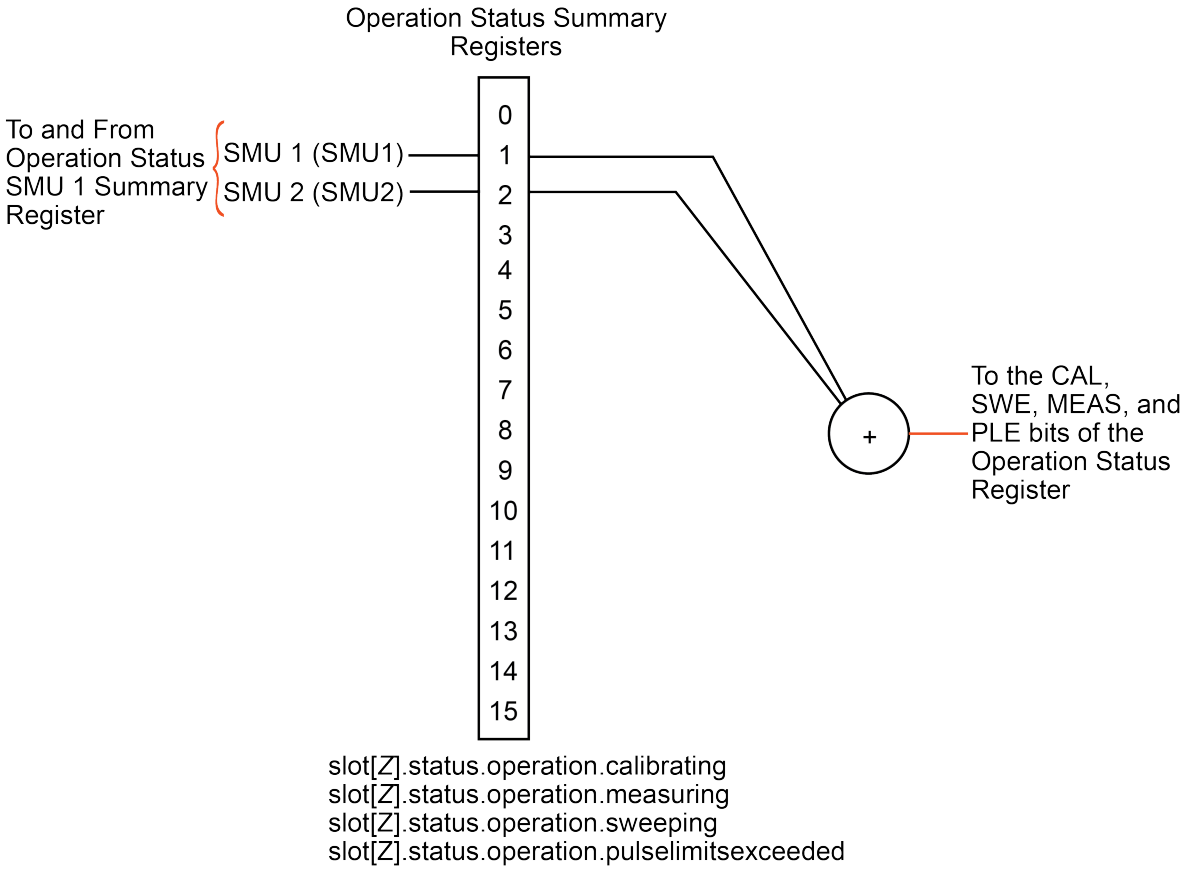
For additional detail, see Program enable and transition registers (on page 636).

For more information on the Operation Status Registers, refer to Status register set contents (on page 634) and the figures in this section.

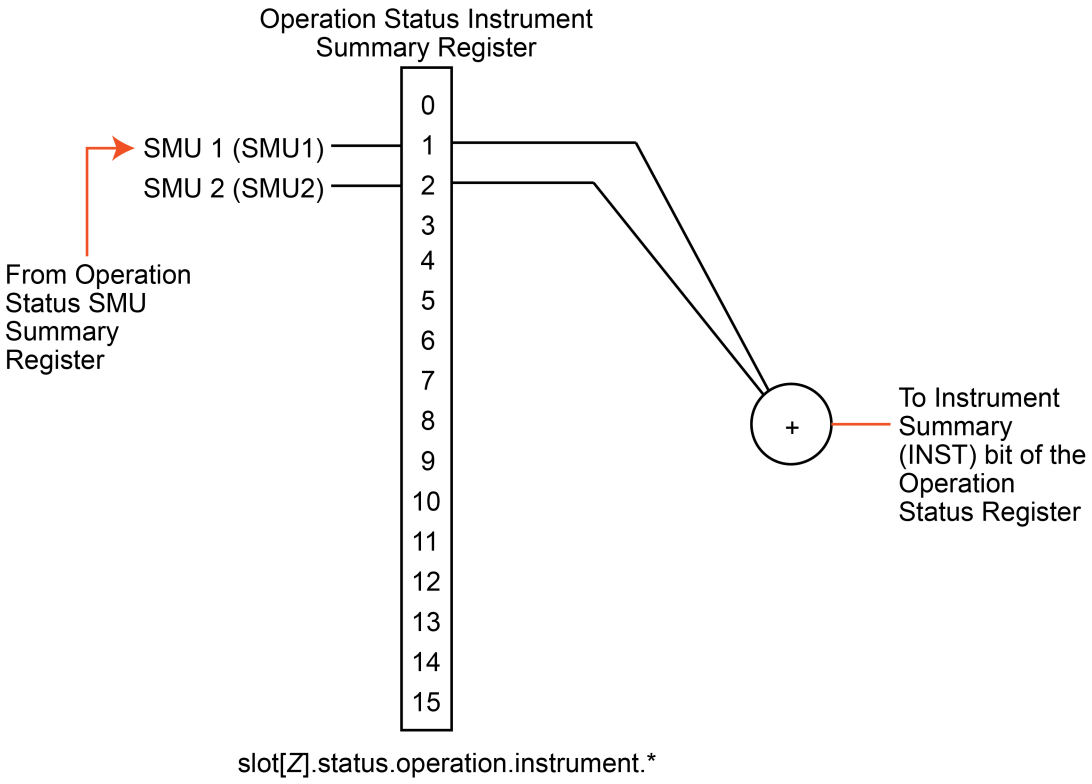
Operation Status Register



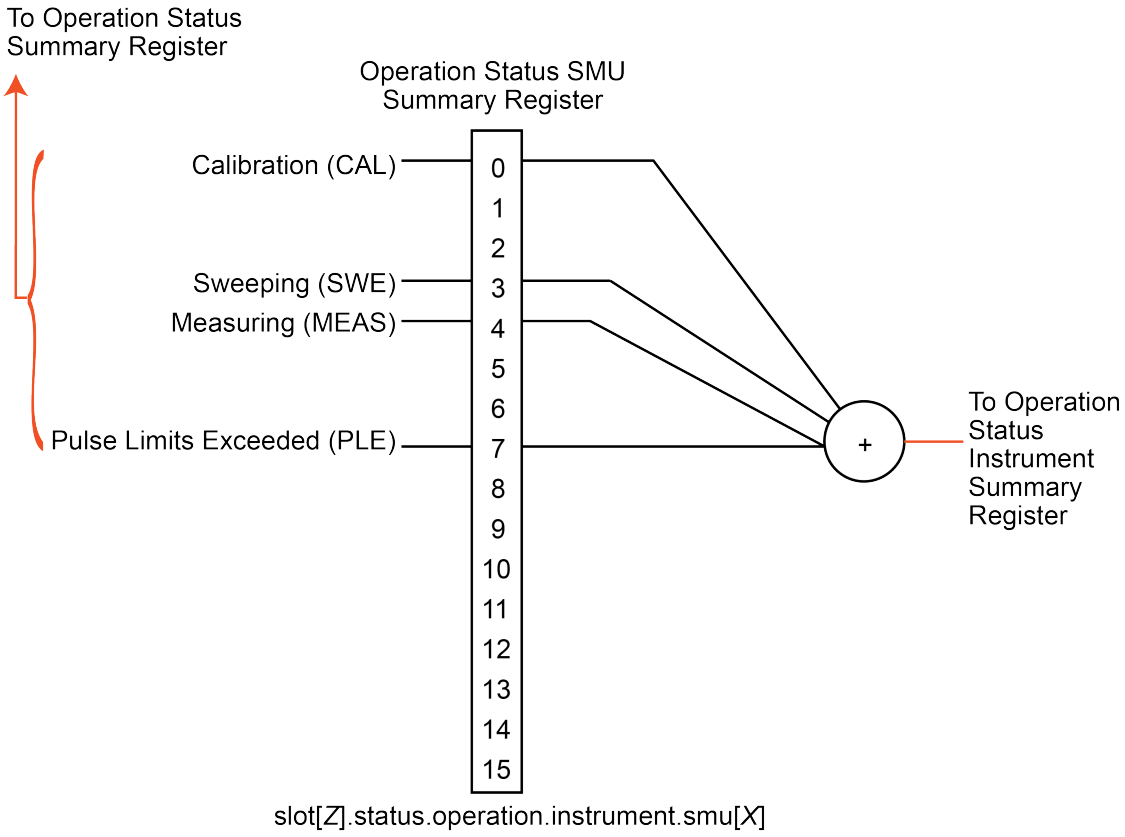
Operation Status Summary Registers



Operation Status Instrument Summary Register



Operation Status SMU Summary Register



MPSU50-2ST status model

Measurement Event Registers

There are several register sets associated with measurement event status. Commands are summarized in Status register set contents (on page 634). Bits can be set by using enumerated values or numeric parameter values. For details, see Program enable and transition registers (on page 636).

For example, either of the following commands sets the reading overflow enable bit:

```
slot[1].status.measurement.enable = slot[1].status.measurement.ROF
slot[1].status.measurement.enable = 128
```

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 Read registers. For example, the following command reads the Measurement Event enable register:

```
print(status.measurement.enable)
```

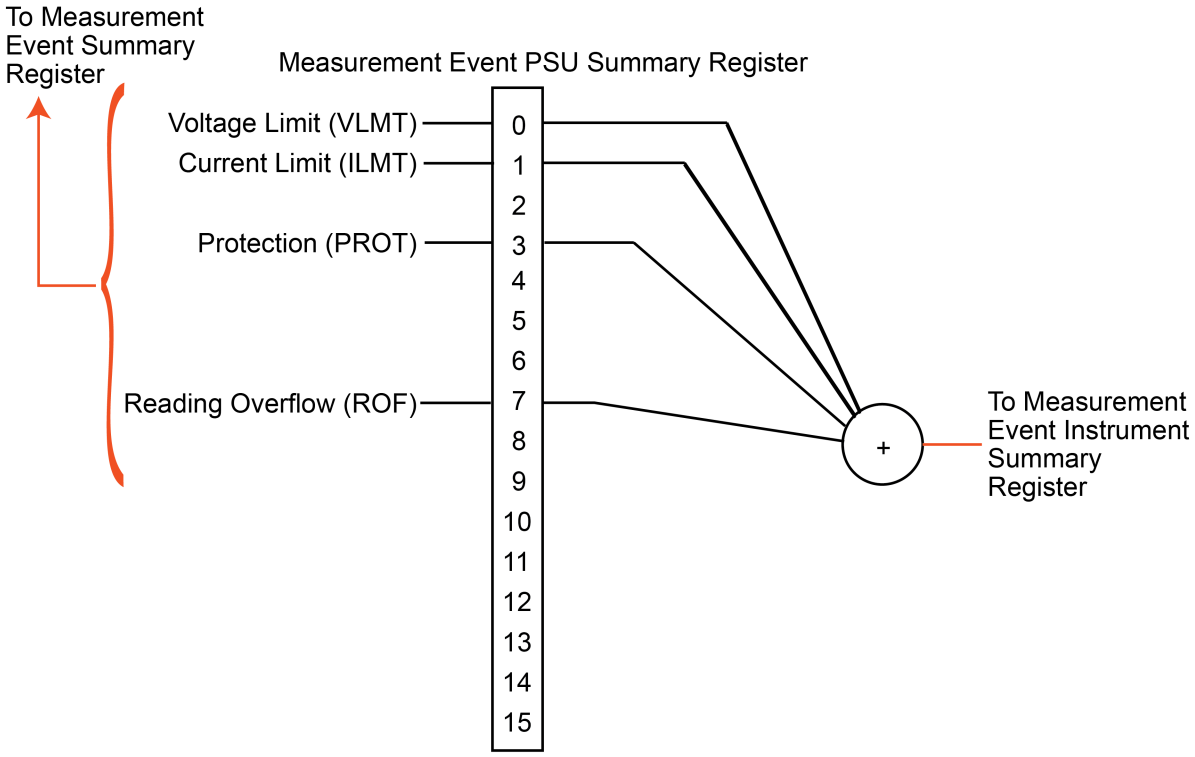
This register set feeds to bit B0 (MSB) of the Status Byte. The bits used in the Measurement Event registers are:

- **Bit B1, Current Limit (ILMT):** Set bit indicates that the current limit was exceeded. This bit is updated only when either a measurement is made or the `slot[Z].smu[X].source.atlimit` attribute is read.
- **Bit B3, Protection (PROT):** Set bit indicates that protection was detected.

- **Bit B7, Reading Overflow (ROF):** Set bit indicates that an overflow reading has been detected.

Commands to program and read the register are summarized in the Status function and attribute summary table. For more information about the Measurement Event registers, refer to Status register set contents (on page 634) and the figures in this section.

Measurement Event PSU Summary Register



slot[*Z*].status.measurement.instrument.psu[*X*]

Measurement Event Summary registers

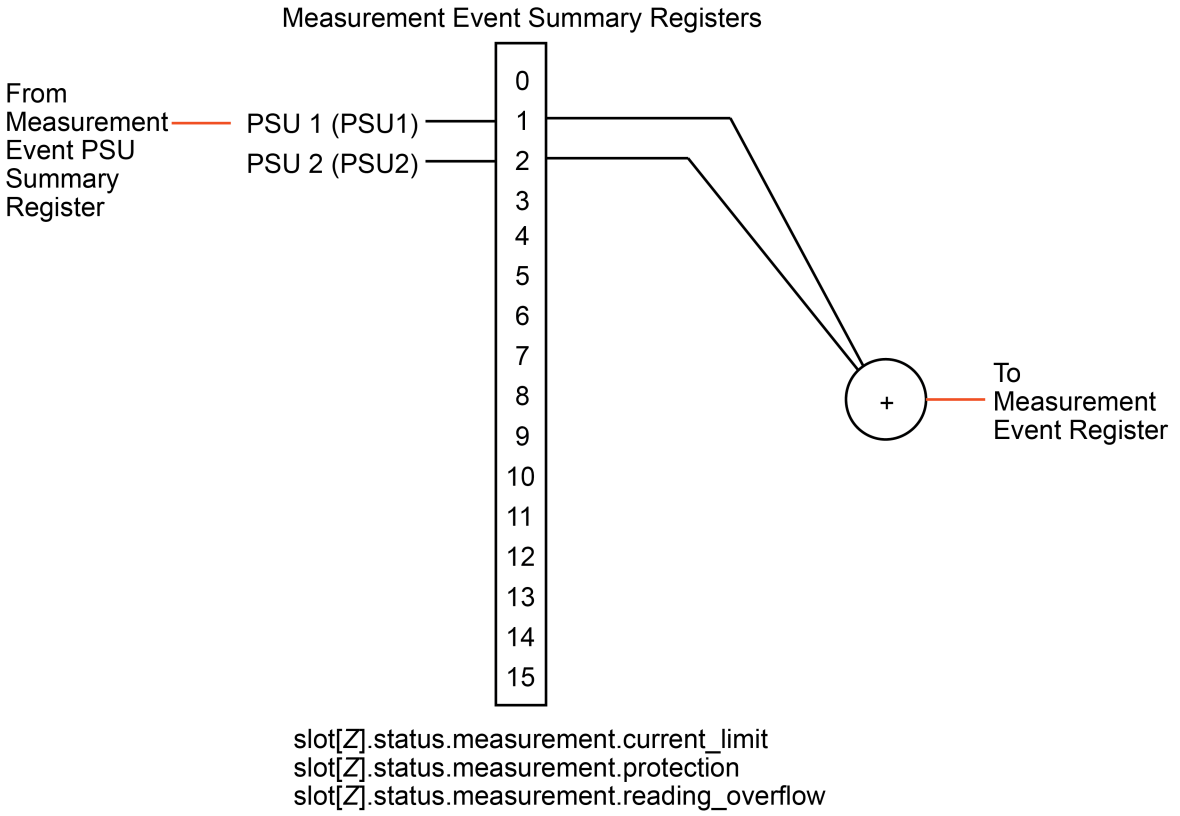
Measurement Event Summary registers are provided for current limit, voltage limit, protection, and reading overflow.

The input to the Measurement Event Summary register comes from the appropriate register in the Measurement Event PSU Summary register:

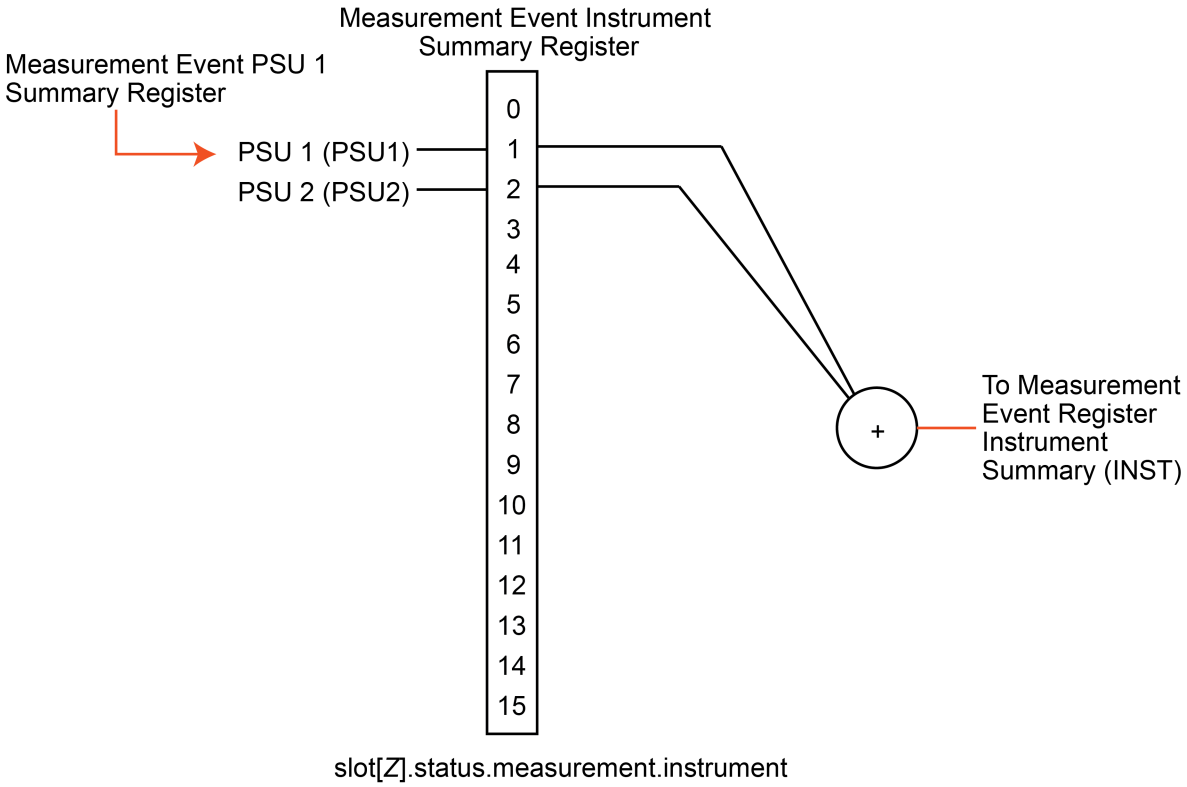
- **Current Limit (ILMT):** Bit 1
- **Protection (PROT):** Bit 3
- **Reading Overflow (ROF):** Bit 7

The output of the Measurement Event Summary register goes to the appropriate register in the Measurement Event register:

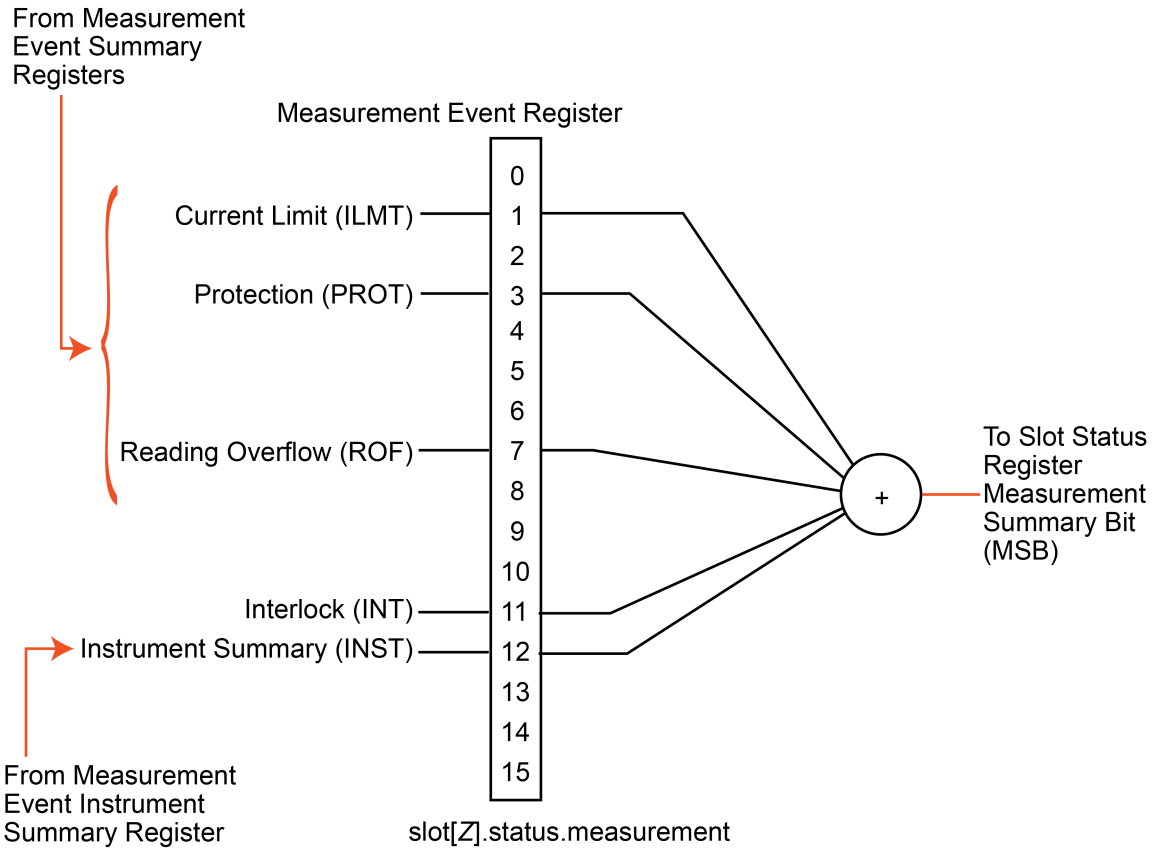
- **Current Limit (ILMT):** Bit 1
- **Protection (PROT):** Bit 3
- **Reading Overflow (ROF):** Bit 7



Measurement Event Instrument Summary Register



Measurement Event Register

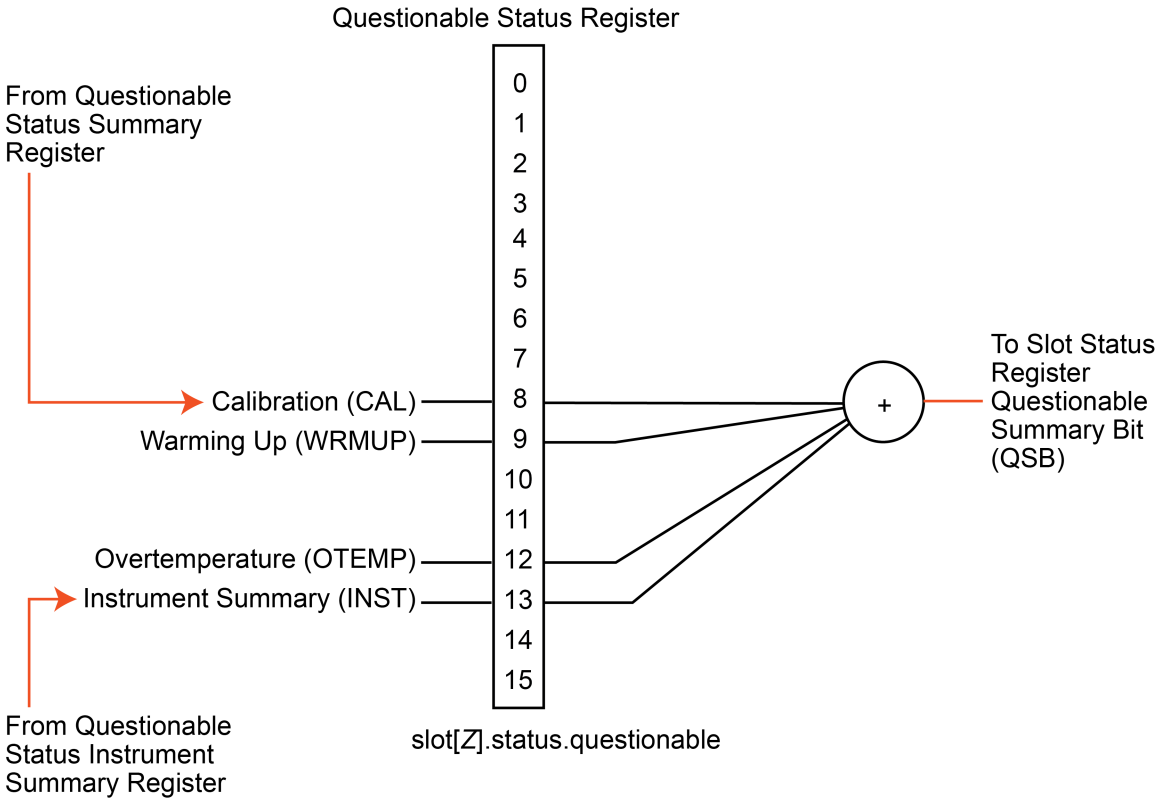


Questionable Status Register

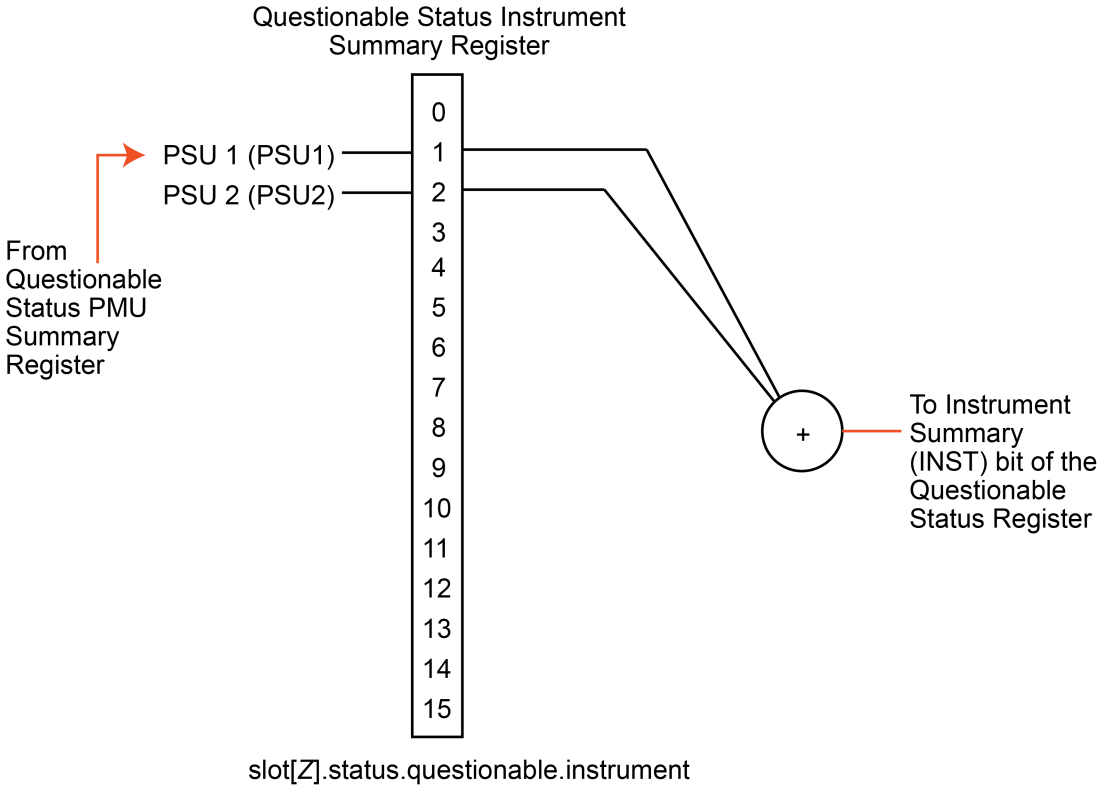
This register set feeds to bit B3 (QSB) of the Slot Status Register. The bits used in the Questionable Status Register set are described as follows:

- **Bit B8, Calibration (CAL):** Set bit indicates that calibration is questionable.
- **Bit B9, Warm Up (WRMUP):** Set bit indicates that the instrument is warmed up and ready for autocalibration.
- **Bit B12, Overtemperature (OTEMP):** Set bit indicates that an overtemperature condition was detected.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that a bit in the Questionable Status Instrument Summary Register is set.

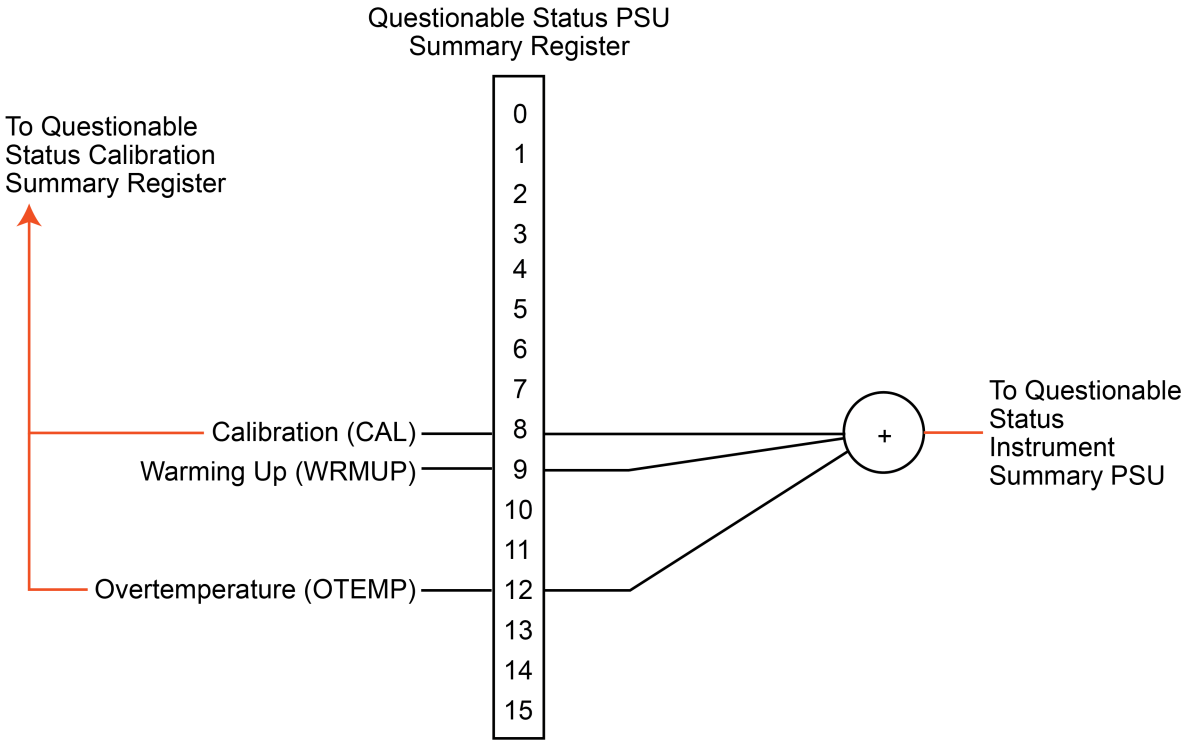
You can read or set the bits using the slot[Z].status.questionable.* (on page 561) command.



Questionable Status Instrument Summary Register

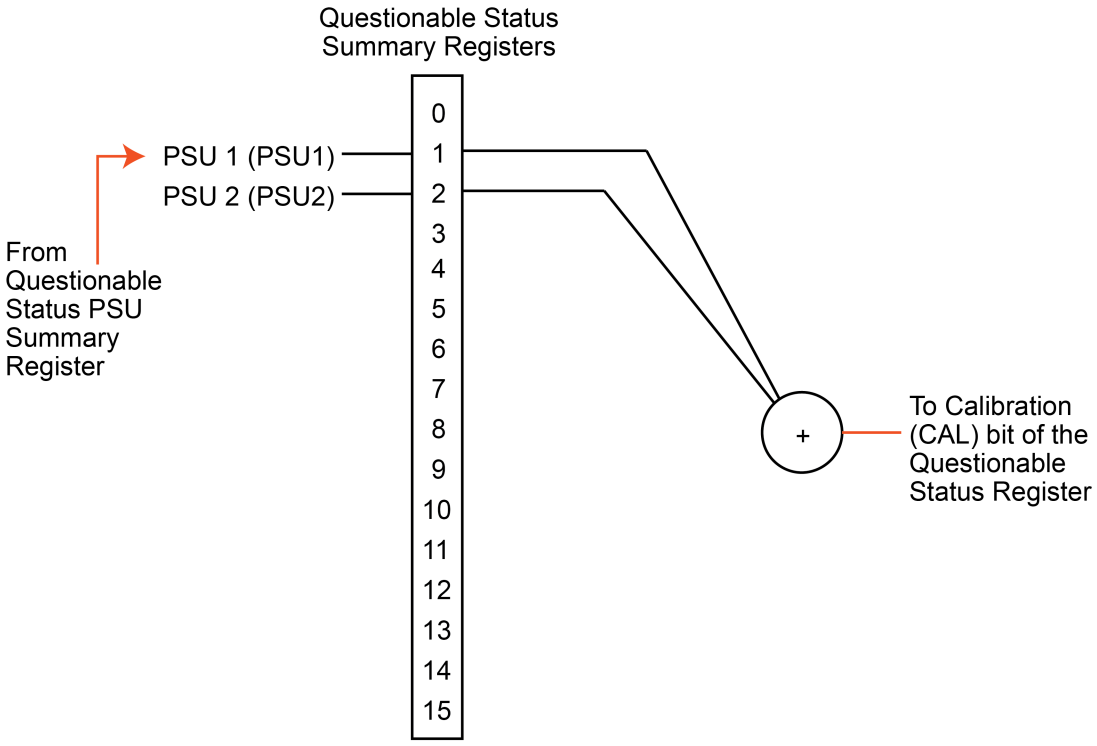


Questionable Status PSU Summary Register



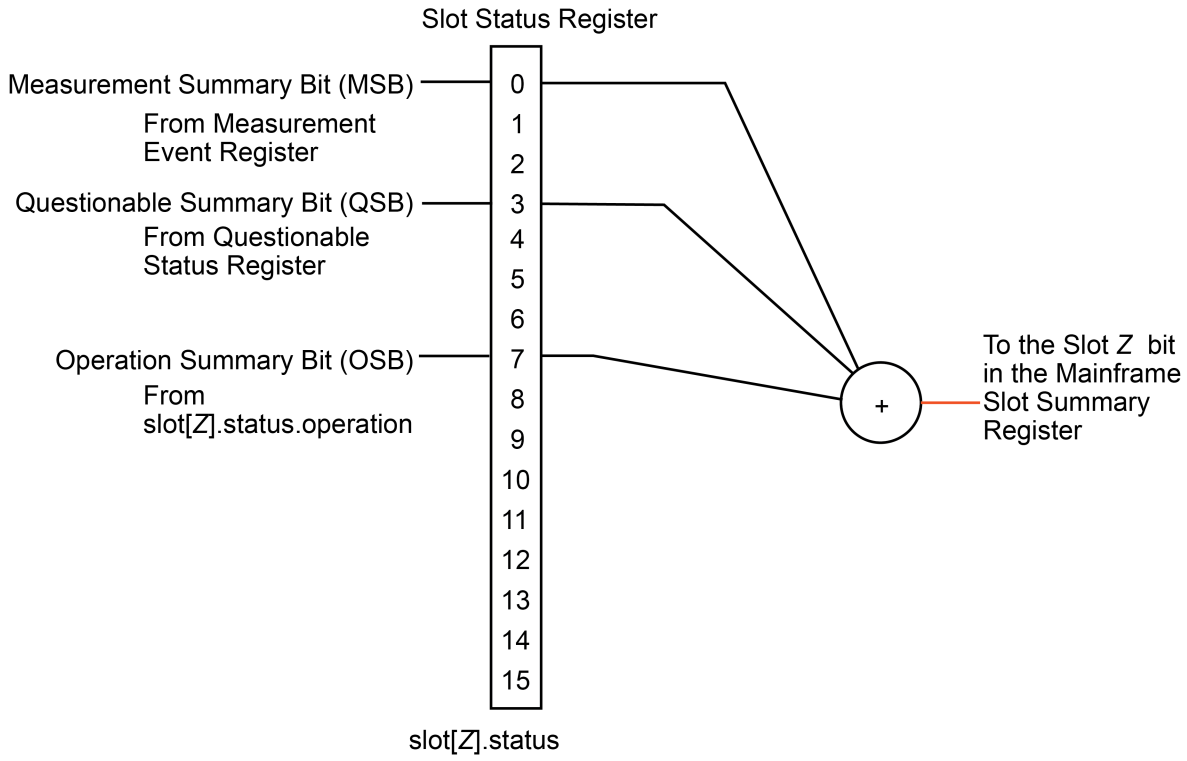
slot[*Z*].status.questionable.instrument.psu[*X*]

Questionable Status Calibration Summary Register



slot[*Z*].status.questionable.calibration
slot[*Z*].status.questionable.over_temperature

Slot Status Register



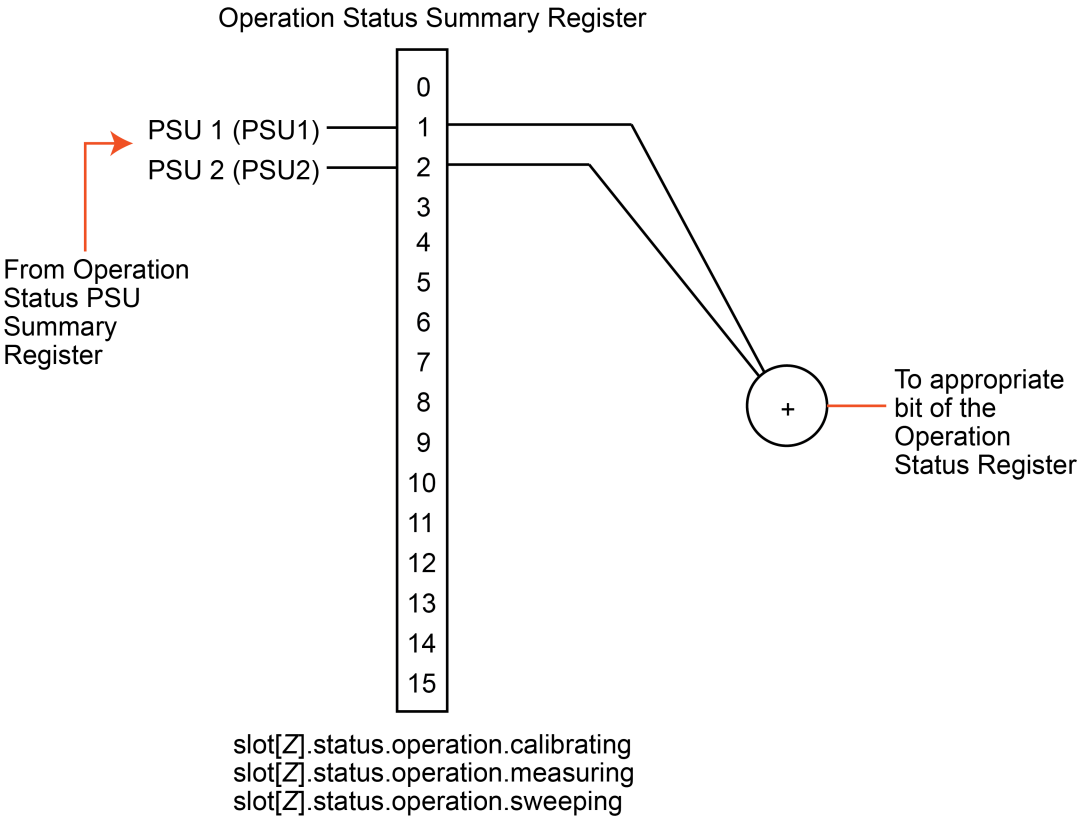
PSU Operation Status Registers

Commands to program and read the PSU Operation Status register set feeds to bit B7 (OSB) of the Status Byte. The bits used in the Operation Status Register set are described as follows:

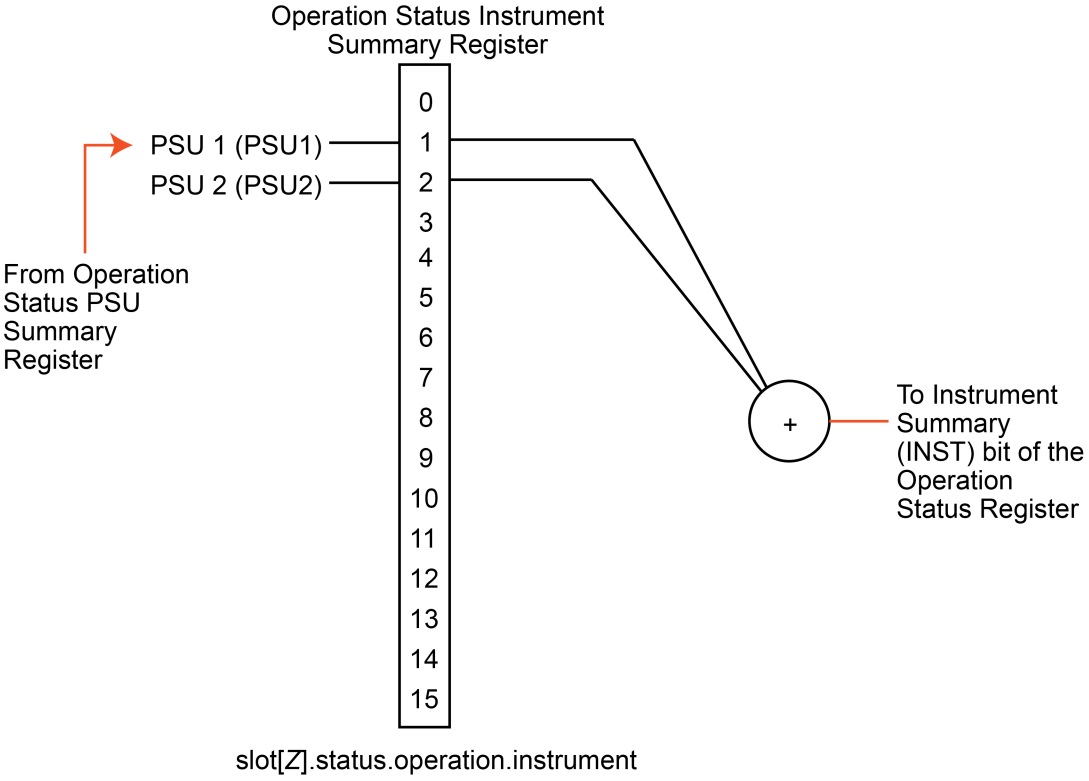
- **Bit B0, Calibrating (CAL):** Set bit indicates that one or more channels are calibrating.
- **Bit B3, Sweeping (SWE):** Set bit indicates that one or more channels are sweeping.
- **Bit B4, Measuring (MEAS):** Bit is set when making an overlapped measurement, but it is not set when making a normal synchronous measurement.
- **Bit B13, Instrument Summary (INST):** Set bit indicates that an enabled bit in the Operation Status Instrument Summary Register is set.

For more information on the Operation Status Registers, refer to Status register set contents (on page 634) and the figures in this section.

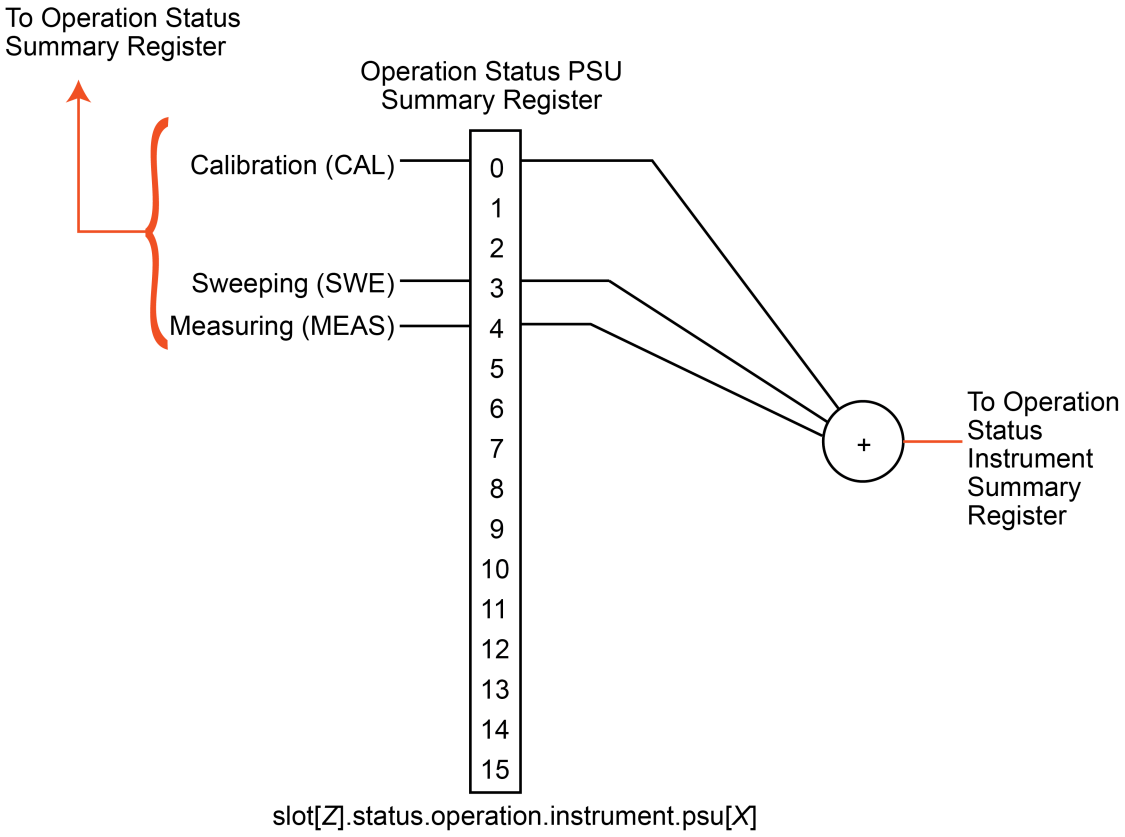
PSU Operation Status Summary Register



PSU Operation Status Instrument Summary Register



Operation Status PSU Summary Register



PSU Operation Status Register