How to Transition Code to TSP from SCPI

APPLICATION NOTE

*CLS	eventlog.clear() status.clear()
*ESE	status.standard.enable
*ESE?	print(status.standard.enable)
*ESR?	print(status.standard.event)
*IDN?	print(localnode.model) print(localnode.serialno) print(localnode.version)
*LANG	No equivalent, accessible through front pa
*LANG?	No equivalent, accessible through front
*OPC	opc()
*OPC?	waitcomplete() print([[1
*RST	reset()
*SRE	status.request_enable

Tektronix®

Introduction

For many years, instrument manufacturers have used "Standard Commands for Programmable Instrumentation" (SCPI) to control programmable test and measurement devices in test systems. The goal of SCPI is to provide a uniform and consistent command set for the control of test and measurement instruments. The same commands and queries control corresponding instrument functions in SCPI equipment, regardless of the manufacturer or the instrument type. However, despite SCPI's intended function as a universal command set, the simple fact that new features are implemented with each new iteration of an instrument means that SCPI command sets will vary between instruments.

The main hurdle to overcome before taking the initiative to use a different command set is learning the new one, however adjusting to the nuances of a new SCPI command implementation is required every time a different instrument is in use, so it is not a hurdle that users are unfamiliar with. The purpose of this document is to introduce Keithley's Test Script Processor (TSP®) command set and scripting language as an alternative to SCPI and traditional instrument programming. First, an overview of SCPI and TSP will be given, followed by several side-by-side examples to show analogous commands between both command sets.

What is TSP?

Keithley's Test Script Processor (TSP) is a flexible hardware/software architecture that allows message-based programming, much like SCPI, with enhanced capabilities for controlling test sequencing/flow, decision-making, and instrument autonomy. TSP-enabled instruments operate like conventional SCPI instruments by responding to a sequence of commands sent by the controller. You can send individual commands to the TSP-enabled instrument the same way you would when using SCPI with any other instrument. Making the switch to TSP will afford you improved throughput, access to additional interfacing options between both the PC and other instruments, and the convenience of autonomous instrumentation when desired.

The use of an on-board Test Script Processor has made it possible to create "smart" instruments, with built-in decisionmaking capabilities, which reduces the need to communicate so frequently with an external controller. This approach to test system design allows smart instrument systems to be much more efficient than those that rely on standard SCPI-based programming. As the number of TSP-based instruments grows, test system developers will have greater flexibility to build test systems with far higher throughput without compromising measurement integrity.

TSP encompasses both the TSP command set and the TSP scripting language. The TSP scripting language is based on Lua version 5.0, and when used together with the TSP command set, allows for logic and subroutines that would normally reside on a PC to run inside the instrument, which reduces the amount of data and number of messages sent over the communications bus by a considerable amount.

Why Use TSP?

While some instruments such as the 2600 Series SMUs can only be interfaced with by using TSP, many Keithley instruments allow the user to choose either SCPI or TSP for remotely controlled instrument applications, which raises the question: Why use TSP if the option of using SCPI is available?

The main reason to choose the TSP command set for remote operations is due to its ability to improve throughput by offloading most processes and calculations onto the instrument. With the computing operations on the instrument side of the communication link, interactions between the host PC and the test equipment can be significantly reduced by eliminating entire reads and writes over the bus from the test sequence. This is especially advantageous when sweeping or when numerous repetitive measurements are made. Instead of sending an entire array of data back to the PC for processing, the instrument can process the array directly and return a single computed answer.

TSP also offers far more flexibility and expandability than the SCPI command set. While offloading some data processing and calculations onto the instrument is one of TSP's main

advantages, that feature is not required to use the TSP command set. TSP can be used in the same manner as SCPI, where a PC controller sends commands to the instrument and the data is returned to the PC to be interpreted by your preferred software tools. The more cohesively your different pieces of test equipment work, the less time you will need to spend implementing workarounds or tricks to get the desired performance outcome, which is why it is good to consider implementing some of TSP's more advanced features and try scripting with TSP at some point.

Like SCPI commands, TSP commands can be sent down from a controller using a programming language such as Python or C++. However, unlike SCPI, TSP offers many embedded features that will allow you to expand the functionality of your test setups. Should you choose to try scripting with TSP, you will acquire the ability to load, store, and run numerous customized test functions on an instrument and broadcast the same to all other connected instruments; and gain access to greatly improved communication throughput via best programming practices.

To learn more about scripting with TSP and other advanced features see the Application Note: *How to Write Scripts for Test Script Processing (TSP).*

Comparing the SCPI and TSP Command Systems

The SCPI command format was developed to be self-descriptive, each command is intended to describe its own function. All SCPI commands have an extended and abbreviated form. The more verbose form aids in user understanding as the commands hold more characters to help better define the action the specific command performs. The advantage of the abbreviated form is that the user can achieve the same outcome sending fewer characters. Every ASCII character sent and received takes time to propagate from sender to receiver. Therefore, fewer characters translates to less communication time.

SCPI Example:

```
:ROUTe[:CHANnel]:CLOSe (@<channelList>)
```

vs.

```
:ROUT:CLOS(@<channelList>)
```

TSP Example:

```
channel.close("channelList")
```

However, in using the truncated SCPI commands, the user sacrifices readability for throughput. TSP language does not have short form commands. However, with scripting, you can have the commands run directly on the instrument itself, removing the need for abbreviated commands as there is no longer any delay caused by bus communication time. Even without full scripting, it is possible to alias TSP commands to any user selected string, allowing even shorter command strings than those of SCPI short commands. Comparatively, a PC-based control program with TSP can yield greater readability with better throughput than when utilizing SCPI.

One of the main differences between the SCPI and TSP command systems is how readings are taken. In the SCPI case, you must send the `:READ?` command to initiate the measurement. The reading is then placed in an output queue in the instrument for retrieval. The control program must then get the reading from the instrument in order to complete the process. This is because SCPI queries must adhere to a message exchange protocol which maintains that a new query or command cannot be sent to the instrument before the result of the previous query is retrieved completely. Otherwise, the "Query Interrupted" error will propagate. This inflexible messaging protocol limits your instrument control options, and can be problematic in some applications.

This is not the case with the TSP command set. Since TSP is built for scripting, readings are stored in variables and can be returned at any time and in any quantity. A measurement is executed by the command `READING = dmm.measure.read()` for digital multimeters and data acquisition systems, or `READING = smu.measure.read()` for source measure units. In both cases, the measurement is stored in the variable `READING`. It is not necessary to return the measurement to the host controller unless it is required, in which case, the `print(READING)` command will return the value. The `READING` variable could be used within a TSP script for other operations, such as limit testing, a math operation, or as part of an overall testing strategy. This is where the power of the TSP functions begins.

The SCPI Instrument Model controls the way instrument functionality is divided among the SCPI command sub-systems. Some sub-systems are only available on certain types of instruments. For example, the ROUTe sub-system is only available on units with switching capabilities or with front and rear terminals. The command sub-systems are generally broken down into the following categories:

1. **CALCulate:** Used for math expressions, limit testing, and statistics.
2. **CALibration:** Configures and controls the calibration operations.
3. **DIGital:** The commands in the Digital subsystem control the digital I/O lines.
4. **DISPlay:** Controls the display of the instruments.
5. **INITiate:** Starts the trigger model and enables/disables continuous triggering.
6. **FORMat:** Selects the data format for transferring readings over the bus.
7. **OUTPut:** Provides information and settings that control the output of the selected source.
8. **ROUTe:** Controls front/rear inputs and channel switching.
9. **SCRipt:** Controls macro or instrument setup scripts.
10. **SENSe:** Configures and controls the measurement functions.
11. **SOURce:** Configure and control the current source and voltage source for SMUs and power supplies.
12. **STATus:** Controls the status registers.
13. **SYSTem:** Contains miscellaneous commands for instrument setup including passwords, beepers, time, etc..
14. **TRACe:** Configures and controls data storage into the buffer.
15. **TRIGger:** Configures the Trigger Model and controls trigger operations.

The TSP command set is a group of predefined functions and attributes that are used to control the instrument. They act as instrument commands that are used in the same manner as SCPI commands used by SCPI instruments. Like SCPI, TSP commands can also be broken down into categories, and not all the categories apply to all instruments. For example, the SMU commands can only be used by source measure units.

1. **Beeper:** Commands used to control the built-in beeper.
2. **Bit:** Used to perform logic operations on one or two binary numbers.
3. **Buffer:** Commands that are used to manipulate the contents and features of the reading buffers.
4. **Channel:** Commands that set the attributes of multiplexer channels available with data loggers and data acquisition systems and monitor their status.
5. **Delay:** Commands the instrument to wait for a specified number of seconds.
6. **Digital I/O:** Used to control the Digital I/O port of the instrument.
7. **Display:** Used to control display messaging on the front panel of the instrument.
8. **DMM:** Controls the measurement operations for DAQs and DMMs.
9. **Error Queue:** Used to read the entries in the error/event queue.
10. **Event Log:** Used to view specific details about triggering and other system events.

11. **Exit:** Used to terminate a script that is presently running.
12. **File:** Used to open and close directories and files, write data, or to read a file from a specified file location.
13. **Format:** Used for data printed with the printnumber and printbuffer commands.
14. **GPIB:** Used to set the GPIB address.
15. **LAN:** Used to review and configure network settings over the remote interface.
16. **LocalNode:** Used to set the power line frequency, control (on/off) prompting, and control (hide/show) error messages on the display.
17. **Operation Complete:** Sets the OPC bit in the status register when all overlapped commands are completed.
18. **Print:** Generates a response message.
19. **PrintBuffer:** Prints data from tables or reading buffer subtables.
20. **Reset:** Used to return an instrument to its default settings.
21. **Scan:** Configures the scan settings for multiplexer modules used with data acquisition systems.
22. **Script:** Commands used to manage TSP scripts stored in the instrument's nonvolatile memory.
23. **Serial:** Used to configure and control the instrument's serial Interface.
24. **Slot:** Commands used to monitor the rear slot for switch or multiplexer cards.
25. **SMU:** Used to control basic source-measure operations of Source Measure Units.
26. **Status:** Controls the status registers.
27. **Timer:** The timer can be used to measure the time it takes to perform various operations.
28. **Trigger:** Used to control triggering.
29. **UserString:** Used to store/retrieve user-defined strings in non-volatile memory.
30. **WaitComplete:** Waits for all overlapped commands to complete.

While the TSP categories are a larger list of command definitions compared to the list of SCPI subsystems, many of the categories in this list contain access to features that are otherwise unavailable when using the SCPI command set. One such advantage over the SCPI command set is the TSP command set's fine-grained control over the instrument's processes, including low-level processes. In comparison, SCPI only allows access to the instrument's high-level features. A large number of categories is also explained by the more straightforward organizational structure. For example, the SCPI SYSTem category contains commands analogous to those in these TSP categories: Beeper, Slot, LAN, Error Queue, Event Log, and GPIB. Several of these categories are individual commands in and of themselves.

Common remote interface commands are commands that have the same general meaning, regardless of the instrument you use them with. The common commands perform operations such as reset, wait-to-continue, and status. They always begin with an asterisk (*) and may include one or more parameters. The command keyword is separated from the first parameter by a blank space. Note that although these commands are essentially the same as those defined by the IEEE Std 488.2 standard, TSP-compatible instruments may not strictly conform to that standard.

If you are using a SCPI remote interface, the commands can be combined. Use a semicolon (;) to separate multiple commands, as shown below:

```
*RST; *CLS; *ESE 32; *OPC?
```

Common commands can be used with TSP-only instruments and instruments currently set to TSP mode. For example, an instrument set to TSP mode will still respond with an ID string when *IDN? is sent to it. However, common commands cannot be used in TSP scripts. Following is a list of the common SCPI commands along with their TSP command set equivalents:

Common Commands with TSP Equivalents

Common SCPI Commands	TSP Equivalent Commands
*CLS	eventlog.clear() status.clear()
*ESE	status.standard.enable
*ESE?	print(status.standard.enable)
*ESR?	print(status.standard.event)
*IDN?	print(localnode.model) print(localnode.serialno) print(localnode.version)
*LANG	No equivalent, accessible through front panel
*LANG?	No equivalent, accessible through front panel
*OPC	opc()
*OPC?	waitcomplete() print([[1]])
*RST	reset()
*SRE	status.request_enable
*SRE?	print(status.request_enable)
*STB?	print(status.condition)
*TRG	trigger.tsplinkout[N].assert()
*TST?	print([[0]])
*WAI	waitcomplete()

How to Switch Code from SCPI to TSP: Examples

Making the switch to the TSP command set from SCPI is a simple matter of swapping analogous commands. While that concept seems daunting, it is not so different from learning the different SCPI command sets for each individual instrument model.

DAQ6510 Example

This example configures a basic scan using a DAQ6510 Data Acquisition and Logging Multimeter. Channels 101 through 109 are set to measure DC voltage at 1 power line cycle with auto range enabled. Resolution is set to 4.5 digits and the scan count is set to 10.

SCPI	TSP
`"*RST"`	`reset()`
`"FUNC 'VOLT:DC', (@101:109)"`	`channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE)`
`"VOLT:NPLC 1, (@101:109)"`	`channel.setdmm("101:109", dmm.ATTR_MEAS_NPLC, 1)`
`"VOLT:RANG:AUTO ON, (@101:109)"`	`channel.setdmm("101:109", dmm.ATTR_MEAS_RANGE_AUTO, dmm.ON)`
`"DISP:VOLT:DIG 4, (@101:109)"`	`channel.setdmm("101:109", dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)`
`"ROUT:SCAN:CRE (@101:109)"`	`scan.create("101:109")`
`"ROUT:SCAN:COUN:SCAN 10"`	`scan.scancount = 10`
`"INIT"`	`trigger.model.initiate()`
`"TRAC:DATA? 1, 90, "defbuffer1", READ"`	`printbuffer(1, 90, defbuffer1.readings)`

Alternatively, the first four TSP commands that apply settings to the channel list can be condensed into a single command by passing all the arguments at once, while the SCPI settings must all be applied separately.

SCPI	TSP
`"*RST"`	`reset()`
`"FUNC 'VOLT:DC', (@101:109)"`	`channel.setdmm("101:109", dmm.ATTR_MEAS_FUNCTION, dmm.FUNC_DC_VOLTAGE, dmm.ATTR_MEAS_NPLC, 1, dmm.ATTR_MEAS_RANGE_AUTO, dmm.ON, dmm.ATTR_MEAS_DIGITS, dmm.DIGITS_4_5)`
`"VOLT:NPLC 1, (@101:109)"`	
`"VOLT:RANG:AUTO ON, (@101:109)"`	
`"DISP:VOLT:DIG 4, (@101:109)"`	
`"ROUT:SCAN:CRE (@101:109)"`	`scan.create("101:109")`
`"ROUT:SCAN:COUN:SCAN 10"`	`scan.scancount = 10`
`"INIT"`	`trigger.model.initiate()`
`"TRAC:DATA? 1, 90, "defbuffer1", READ"`	`printbuffer(1, 90, defbuffer1.readings)`

SMU 2450 Example

This example utilizes a 2450 SourceMeter® source measure unit to produce an I-V sweep characterization of a solar cell. The voltage is swept from 0 V to 0.55 V in 56 steps. The resulting solar cell current is measured. The current and voltage measurements are stored in a default data buffer (`defbuffer1`). Finally, the delta buffer is returned.

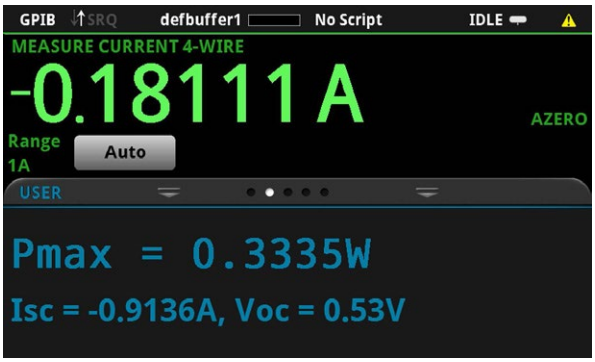
SCPI	TSP
`"*RST"`	`reset()`
`"SENS:FUNC 'CURR'"`	`smu.measure.func = smu.FUNC_DC_CURRENT`
`"SENS:CURR:RANG:AUTO ON"`	`smu.measure.autorange = smu.ON`
`"SENS:CURR:RSEN ON"`	`smu.measure.sense = smu.SENSE_4WIRE`
`"SOUR:FUNC VOLT"`	`smu.source.func = smu.FUNC_DC_VOLTAGE`
`"SOUR:VOLT:RANG 2"`	`smu.source.range = 2`
`"SOUR:VOLT:ILIM 1"`	`smu.source.ilimit.level = 1`
`"SOUR:SWE:VOLT:LIN 0, 0.55, 56, 0.1"`	`smu.source.sweeplinear("SolarCell", 0, 0.55, 56, 0.1)`
`"INIT"`	`trigger.model.initiate()`
`"*WAI"`	`waitcomplete()`
`"TRAC:DATA? 1, 56, "defbuffer1", SOUR, READ"`	`printbuffer(1, 56, defbuffer1.sourcevalues, defbuffer1.readings)`

This example can be taken a step further by using these basic TSP commands in conjunction with the full power of the TSP scripting language. The Test Script Processor in enabled instruments uses scripting with the TSP command set. This allows for data interpretation to be handled locally by the instrument, as opposed to remotely by a controlling PC running programs such as Microsoft Excel. The calculations required to identify key points of data can be done by the instrument, and displayed on the front panel or returned to an external computer.

Like the previous example, the instrument here is programmed to produce an I-V sweep characterization of a solar cell where the voltage is swept from 0 V to 0.55 V in 56 steps. But this example enhances functionality by additionally displaying calculated data and custom text on the instrument front panel using the `display.changescreen` and `display.settext` commands. After the test is finished, the front panel display will indicate the maximum power (`Pmax`), the short circuit current (`ISC`), and the open circuit voltage (`VOC`).

TSP

```
--Define the number of points in the sweep.
num = 56
--Reset the instrument and clear the buffer.
reset()
--Set the source and measure functions.
smu.measure.func = smu.FUNC_DC_CURRENT
smu.source.func = smu.FUNC_DC_VOLTAGE
--Measurement settings.
smu.measure.terminals = smu.TERMINALS_FRONT
smu.measure.sense = smu.SENSE_4WIRE
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
--Source settings.
smu.source.highc = smu.OFF
smu.source.range = 2
smu.source.readback = smu.ON
smu.source.ilimit.level = 1
smu.source.sweeplinear("SolarCell", 0, 0.55, num, 0.1)
--Start the trigger model and wait for it to complete.
trigger.model.initiate()
waitcomplete()
--Define initial values.
voltage = defbuffer1.sourcevalues
current = defbuffer1
isc = current[1]
mincurr = current[1]
imax = current[1]
voc = voltage[1]
vmax = voltage[1]
pmax = voltage[1]*current[1]
--Calculate values: maximum power, maximum current, maximum voltage, open circuit voltage,
short circuit current.
for i = 1, num do
   print(voltage[i],current[i],voltage[i]*current[i])
   if (voltage[i]*current[i] > pmax) then
      pmax = voltage[i]*current[i]
      imax = current[i]
      vmax = voltage[i]
   end
   if math.abs(current[i]) < math.abs(mincurr) then
      voc = voltage[i]
   end
end
pmax = math.abs(pmax)
imax = math.abs(imax)
--Display values on the front panel.
display.changescreen(display.SCREEN_USER_SWIPE)
display.settext(display.TEXT1, string.format("Pmax = %.4fW", pmax))
display.settext(display.TEXT2, string.format("Isc = %.4fA, Voc = %.2fV", isc, voc))
```



The display of the 2450 after the script has completed showing the output values in a custom display.

Conclusion

Whether you are using the TSP command set as a replacement for SCPI or using the full TSP language as a powerful scripting tool, TSP can help improve test throughput and increase the overall functionality of your instruments. TSP affords the user a multitude of advantages over SCPI, including the ability to return multiple readings at once, improved throughput, and better readability. Flexibility is a key asset of the TSP command set, allowing the user to tune their experience to their specific needs. TSP can be used in a similar manner to SCPI by being run from a controlling PC with TSP commands acting as analogous replacements to SCPI commands. It can be used to write scripts that are run locally on the instrument, or to manage large networks of connected instruments. To learn more about scripting with TSP in order to get the most out of your TSP compatible instruments, visit tek.com/keithley.

Contact Information:

Australia 1 800 709 465
Austria* 00800 2255 4835
Balkans, Israel, South Africa and other ISE Countries +41 52 675 3777
Belgium* 00800 2255 4835
Brazil +55 (11) 3759 7627
Canada 1 800 833 9200
Central East Europe / Baltics +41 52 675 3777
Central Europe / Greece +41 52 675 3777
Denmark +45 80 88 1401
Finland +41 52 675 3777
France* 00800 2255 4835
Germany* 00800 2255 4835
Hong Kong 400 820 5835
India 000 800 650 1835
Indonesia 007 803 601 5249
Italy 00800 2255 4835
Japan 81 (3) 6714 3086
Luxembourg +41 52 675 3777
Malaysia 1 800 22 55835
Mexico, Central/South America and Caribbean 52 (55) 56 04 50 90
Middle East, Asia, and North Africa +41 52 675 3777
The Netherlands* 00800 2255 4835
New Zealand 0800 800 238
Norway 800 16098
People's Republic of China 400 820 5835
Philippines 1 800 1601 0077
Poland +41 52 675 3777
Portugal 80 08 12370
Republic of Korea +82 2 565 1455
Russia / CIS +7 (495) 6647564
Singapore 800 6011 473
South Africa +41 52 675 3777
Spain* 00800 2255 4835
Sweden* 00800 2255 4835
Switzerland* 00800 2255 4835
Taiwan 886 (2) 2656 6688
Thailand 1 800 011 931
United Kingdom / Ireland* 00800 2255 4835
USA 1 800 833 9200
Vietnam 12060128

*** European toll-free number. If not accessible, call:** +41 52 675 3777

Rev. 02.2018

KEITHLEY
A Tektronix Company

Find more valuable resources at TEK.COM

031121 SBG 1KW-61539-0