Tektronix®

Getting Started with the MP5000 Series Modular Precision Test System and Test Automation

TECHNICAL BRIEF


```
].source.func = slot[1].smu[2
2].source.rangei = 1000e-3')
u[2].source.leveli = 0')
mu[2].source.limitv = 6')

lot[1].smu[2].measure.autorangei = 1')

('slot[1].smu[2].sense = slot[1].smu[2].SENS
e stop current / (number of sweep points -1)
 300e-3 / (31 - 1)
Output On
.write('slot[1].smu[2].source.output = 1')

for each sweep point, calculate the source level and
```

Introduction

Validation Engineers, Production Test Engineers, and System Integrators are often tasked with developing automated test equipment (ATE) systems, which are widely used for testing various semiconductor devices, such as transistors, diodes, MOSFETs and ASICs.

Automated tests help identify defects, verify device performance, and ensure the quality of components, especially in the semiconductor, aerospace, and defense industries where reliability is critical.

The MP5000 Series is a modular ATE system from Tektronix currently featuring DC power supply and source measure unit modules. These modules can be mixed and matched for a variety of testing applications. Like the 26xx series source measure units (SMUs), the MP5000 Series mainframe and modules accept the TSP™ (Test Script Processor) command set.

TSP is a flexible hardware/software architecture that allows for message-based programming. It is similar to SCPI, while adding enhanced capabilities for controlling test sequencing/flow, decision-making, and instrument autonomy. TSP-enabled instruments can operate like conventional SCPI instruments by responding to a sequence of commands sent by the PC.

In this guide, we will go over:

1. Familiarizing yourself with the instrument's TSP command set
2. How to sequence commands
3. Building a test
4. Integrating the test into your test environment

Learning the TSP commands

Before automating your MP5000 Series, it is important to first familiarize yourself with the instrument and its TSP command set.

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 some instruments. Like SCPI, TSP commands can also be broken down into categories, and not all the categories apply to all instruments. The following demonstrates the difference in syntax between traditional SCPI commands and TSP commands using a 2461 SMU.

SCPI EXAMPLE:	TSP EXAMPLE:
`*RST`	`reset()`
`SOURce:FUNCtion VOLTage`	`smu.source.func = smu.FUNC_DC_VOLTAGE`

The TSP commands for controlling the MP5000 Series can be found in the MP5000 Series Programmer's Manual. The command structure for the MP5000 Series requires the user to specify both the slot and channel of the target instrument within the MP5103 mainframe as part of the command structure.

TSP MSMU60-2 Example:

```
reset()
slot[1].smu[2].source.func = slot[1].smu[2].FUNC_DC_VOLTAGE
```

These commands can be shortened for speed and readability via aliasing. An alias is a variable that contains a portion of the table structure of the command. By creating an alias, you can shorten and customize TSP commands to your application.

Alasing Example:

```
gateSMU = slot[1].smu[2]
gateSMU.source.func = gateSMU.FUNC_DC_VOLTAGE
```

Sequencing commands

The first step to automating your new MP5000 Modular Precision Test System is to create a simple test routine by sequencing TSP commands together. This can be done by simply sending individual TSP commands via any program or language just like using SCPI commands. Popular options for this method are Python and C#.

First, you will need to establish a connection with the instrument. This can be done using VISA, as illustrated by the following Python example:

```
import pyvisa
rm = pyvisa.ResourceManager()
inst = rm.open_resource('TCPIP0::192.168.0.2::hislip0::INSTR')
```

Next, send the necessary commands to configure your test settings. Once the parameters are set, you can turn on the instrument output and use programming logic to achieve the desired behavior.

Below is a programming example using Python to send TSP commands to an MP5000 MSMU60-2 to perform a simple current sweep:

```
inst.write('reset()')
#Source Settings
inst.write('slot[1].smu[2].source.func = slot[1].smu[2].FUNC_DC_CURRENT')
inst.write('slot[1].smu[2].source.rangei = 1000e-3')
inst.write('slot[1].smu[2].source.leveli = 0')
inst.write('slot[1].smu[2].source.limitv = 6')
#Measure Settings
inst.write('slot[1].smu[2].measure.rangev = 6')
inst.write('slot[1].smu[2].measure.rangei = 1000e-3')
inst.write('slot[1].smu[2].measure.nplc = 1')
inst.write('slot[1].smu[2].measure.autorangei = 1')
#Sense Mode
inst.write('slot[1].smu[2].sense = slot[1].smu[2].SENSE_2WIRE')
#Calculate stop current / (number of sweep points -1)
delta = 300e-3 / (31 - 1)
#Turn Output On
inst.write('slot[1].smu[2].source.output = 1')

#for each sweep point, calculate the source level and then take an iv
measurement
```

```
for j in range(1, 31):
    inst.write('slot[1].smu[2].source.leveli =' str((j-1) * delta))
    inst.write('slot[1].smu[2].measure.iv(slot[1].smu[2].defbuffer1,
slot[1].smu[2].defbuffer2)')
#Turn Output Off
inst.write('slot[1].smu[2].source.output = 1')
```

For users who want additional abstraction from the TSP commands or who may use many different instruments, using a driver can simplify the code writing process. Tektronix has Python drivers and IVI drivers available for use with the MP5000. These drivers can be used alongside any IDE or code editor, like Visual Studio Code.

Tektronix's Python driver library tm_devices is a device management package that includes a multitude of commands and functions to help users easily automate tests on a broad range of Tektronix instrumentation using Python and supports code-completion aids. This driver package makes coding and test automation simple and easy for engineers with software skills of any level. Installation is also simple and uses pip, Python's package-management system.

To install the library, open a terminal and enter:

```
pip install tm_devices
```

Below are some usage examples for using the tm_devices Python driver to automate an MP5000 MPSU50-2ST:

```
from typing import cast, TYPE_CHECKING

from tm_devices import DeviceManager
from tm_devices.drivers import MP5103

if TYPE_CHECKING:
    from tm_devices.commands import PSU50STCommands

with DeviceManager(verbose=True) as device_manager:
    # Add a mainframe to the device manager and access its commands.
    mainframe: MP5103 = device_manager.add_mf("0.0.0.0")
    # Some examples demonstrating the usage of mainframe level commands.
    mf_model = mainframe.commands.localnode.model
    value = mainframe.commands.eventlog.count

    # Get access to the psu module command object available in third slot of
the mainframe.
    modular_psu = cast("PSU50STCommands",
mainframe.get_module_commands_psu(slot=3))

    # Some examples demonstrating the usage of module level commands.
    # Get the psu model and version
    psu_model = modular_psu.model
    psu_version = modular_psu.version
    modular_psu.firmware.verify()

    # Some examples demonstrating the usage of channel level commands.
```

```
# Set the measurement aperture in seconds
modular_psu.psu[1].measure.count = 5
# Enable the source output
modular_psu.psu[2].source.output = 1
# Set the offset value used for voltage measurements
rel_value = modular_psu.psu[1].measure.rel.levelv
# Create a reference to the default buffer
my_buffer = modular_psu.psu[1].defbuffer1
# Read the value in the specified reading buffer
# Measure the voltage on channel 1 of the PSU
voltage_value = modular_psu.psu[1].measure.v()
```

Building a Test

When it comes to test development with the MP5000 there are two options: traditional test development where the PC software controls the test execution or fully utilizing TSP as both a command set and programming language by writing TSP scripts.

Scripting is an integral function of TSP that allows users to have direct, automatic control of their instrument without the need for an external computer to do the processing, similar to scripting for an embedded device. TSP scripts can accomplish complex tasks ranging from changing a sourced value based on the most recent measurement to synchronizing trigger sweeps across multiple instruments, or even simple tasks like applying a mathematical formula to readings in a buffer. Scripting turns the instrument into a powerful edge solution that can make decisions on the fly by reducing communication overhead and simplifying data analysis.

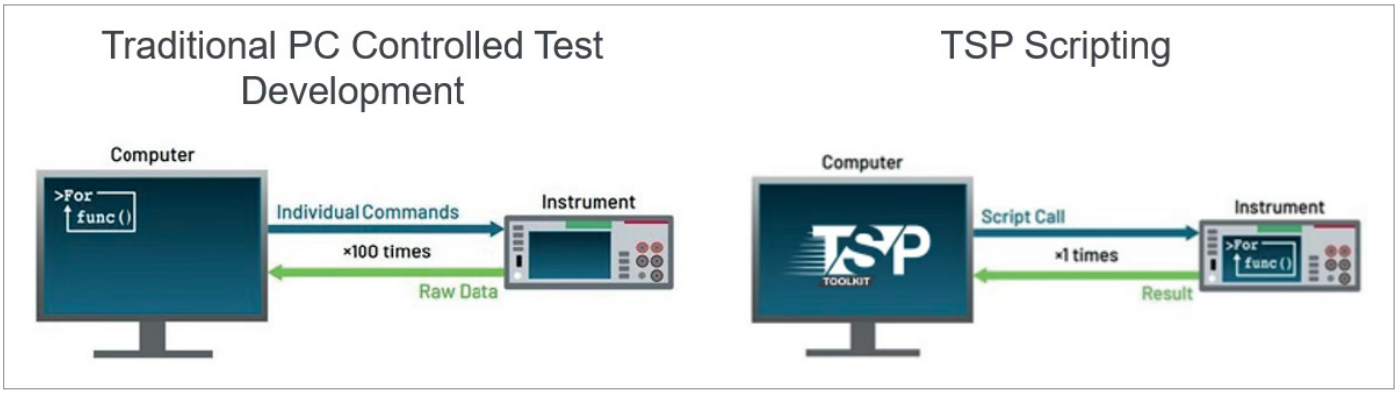

Figure 1: A visual representation of traditional test development on the PC vs a TSP script running on the instrument.

Writing TSP scripts can be done in a text editor or by using Tektronix's TSP Toolkit, a Visual Studio Code extension. This extension includes features that improve the development experience including syntax highlighting, autocompletion with in-line TSP command help, a full-fledged debugger and much more.

TSP scripts can be saved onto the MP5000 to run locally or run from a connected terminal using TSP Toolkit via a sockets or VISA connection.

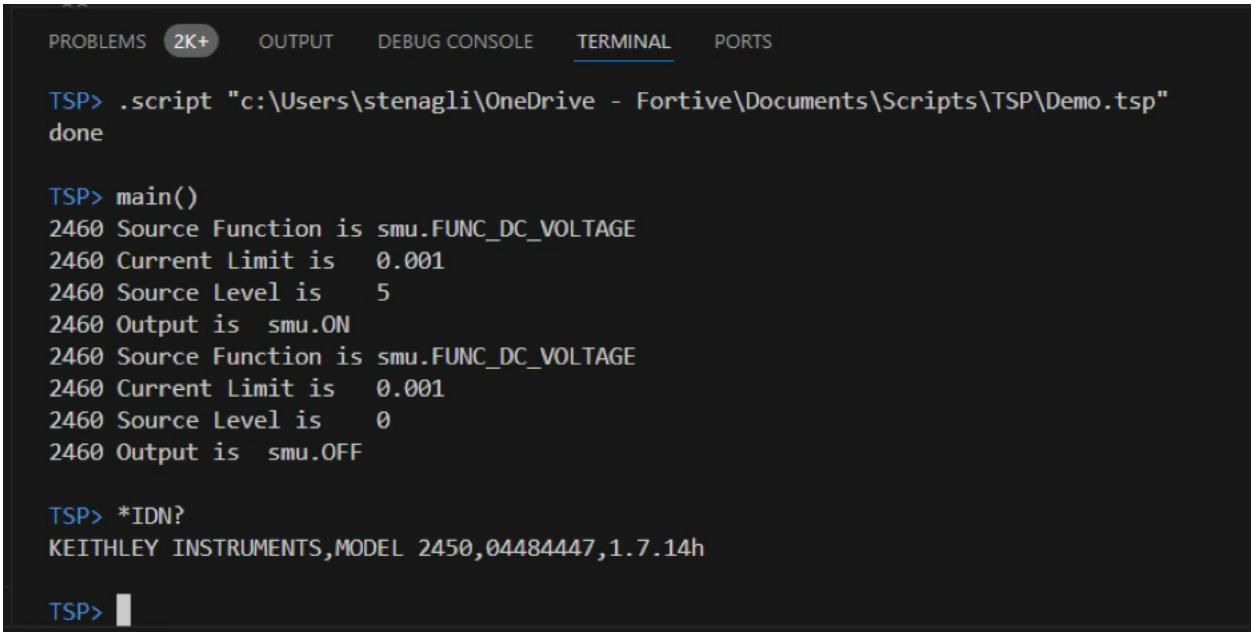

Figure 2: TSP scripts, function calls, and commands running in the TSP Toolkit terminal

Important rules regarding TSP scripts:

- Each script must have a unique name that cannot start with a number.
- Script names must not contain spaces.
- Script names must be unique. If you load a new script with the same name as an existing script, an error event message is generated. You must delete the existing script before you create a new script with the same name.
- If you revise a script and save it to the instrument with a new name, the previously loaded script remains in the instrument with the original name.
- You can save scripts to nonvolatile memory in the instrument. Saving a script to nonvolatile memory allows the instrument to be turned off without losing the script.

Below is a simple TSP script example that uses an MP5000 MSMU60-2 to perform a simple current sweep:

```
reset()
-- Source Settings
slot[1].smu[2].source.func = slot[1].smu[2].FUNC_DC_CURRENT
slot[1].smu[2].source.rangei = 1000e-3
slot[1].smu[2].source.leveli = 0
slot[1].smu[2].source.limitv = 6
-- Measure Settings
slot[1].smu[2].measure.rangev = 6
slot[1].smu[2].measure.rangei = 1000e-3
slot[1].smu[2].measure.nplc = 1
slot[1].smu[2].measure.autorangei = 1
```

```
--Sense Mode
slot[1].smu[2].sense = slot[1].smu[2].SENSE_2WIRE
--Calculate stop current / (number of sweep points -1)
local delta = 300e-3 / (31 - 1)
--Turn Output On
slot[1].smu[2].source.output = 1

--for each sweep point, calculate the source level and then take an iv
measurement
for j = 1, 31 , 1 do
    slot[1].smu[2].source.leveli = ((j-1) * delta)
    slot[1].smu[2].measure.iv(slot[1].smu[2].defbuffer1,
slot[1].smu[2].defbuffer2)
end
```

TSP commands can be grouped together and combined with other programming logic to create a function. This becomes a script, which can be interpreted by the instrument as though it were a single TSP command!

Integrating the test into your test environment

You can use Python or another programming language to call TSP scripts. This capability means that you can leverage partial or fully developed code in another programming language and avoid completely refactoring your code base, while taking advantage of improved throughput, synchronization and triggering with TSP. TSP Scripts are easy to write and debug using TSP Toolkit, and the rest of your framework can be edited using the corresponding VS Code extension for convenience.

A TSP script that has been loaded onto the instrument's memory can be executed from a single line of Python code:

```
import pyvisa
rm = pyvisa.ResourceManager()
inst = rm.open_resource('TCPIP0::0.0.0.0::inst0::INSTR')

inst.write('SimpleIVTest()')
```

Another consideration is getting the data off the instrument. You can print buffered data directly to the TSP Toolkit terminal with TSP commands:

```
--for each sweep point, calculate the source level, take an iv measurement, and
then print the values to the terminal
print("Current\t\t Voltage")
for j = 1, 31 , 1 do
    slot[1].smu[2].source.leveli = 0 + ((j-1) * delta)
    slot[1].smu[2].measure.iv(slot[1].smu[2].defbuffer1,
slot[1].smu[2].defbuffer2)
    print(slot[1].smu[2].defbuffer1[j],slot[1].smu[2].defbuffer2[j])
end
```

These commands can also be used in Python to accomplish the same task:

```
#for each sweep point, calculate the source level, take an iv measurement, and
then print the values to the terminal
print("Current\t\t Voltage")
for j in range(1, 31):
    inst.write('slot[1].smu[2].source.leveli =' str((j-1) * delta))
    inst.write('slot[1].smu[2].measure.iv(slot[1].smu[2].defbuffer1,
slot[1].smu[2].defbuffer2)')

inst.query('print(slot[1].smu[2].defbuffer1[j],slot[1].smu[2].defbuffer2[j])')
```

Conclusion

The MP5000 Modular Precision Test System, combined with the flexibility of TSP scripting and driver support, provides engineers with a powerful and scalable platform for automated test development. By learning the TSP command structure, sequencing commands, building reusable scripts, and integrating them into existing environments, users can streamline workflows, reduce communication overhead, and achieve faster, more reliable test results. Whether leveraging Python drivers for simplicity or fully embracing embedded TSP scripting, the MP5000 enables engineers to create efficient, adaptable, and future-ready test solutions that support a wide range of semiconductor and electronic device validation needs.

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) 3530-8901
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) 88 69 35 25
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. 02.2022

Find more valuable resources at TEK.COM

091925 SBG 1KW-74194-0