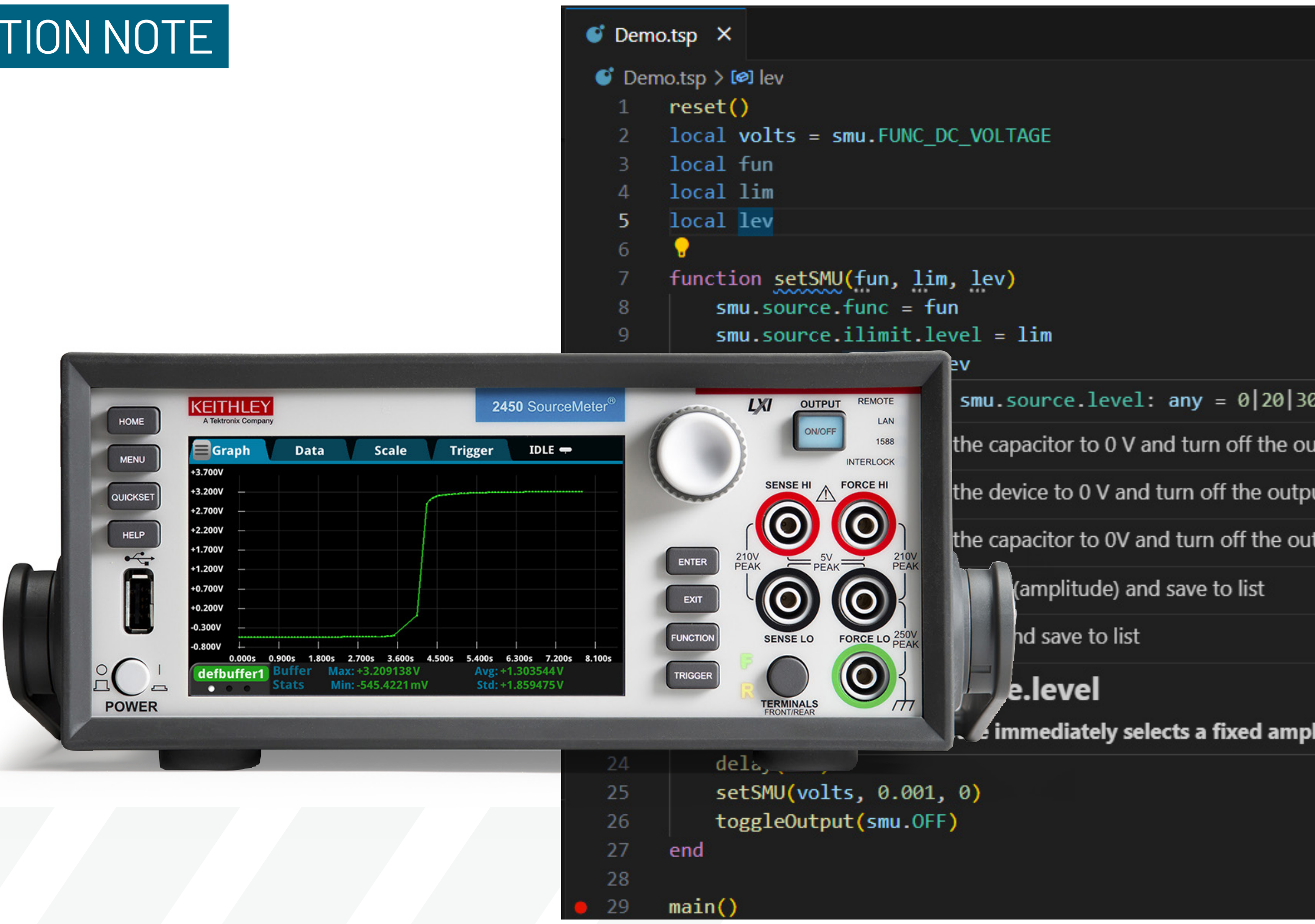
Harness the Power of TSP™ Toolkit Software

APPLICATION NOTE



```
reset()
local volts = smu.FUNC_DC_VOLTAGE
local fun
local lim
local lev

function setSMU(fun, lim, lev)
    smu.source.func = fun
    smu.source.ilimit.level = lim
...
    delay
    setSMU(volts, 0.001, 0)
    toggleOutput(smu.OFF)
end

main()
```

KEITHLEY
A Tektronix Company

Introduction

In an industry where rapid test development is crucial, the need for effective automation and easy code development has never been more pronounced. As businesses strive to enhance their quality while reducing time to market, the right tools can make all the difference. Enter Keithley TSP Toolkit, a new scripting environment that makes adopting Keithley's Test Script Processor (TSP) language and leveraging the benefits of on-instrument scripting more straightforward than ever.

In this application note, we'll define TSP and TSP Toolkit and share some tips and tricks to help you leverage these tools to increase your test throughput.

What is TSP?

TSP is a unique instrument automation command set and programming language. TSP-enabled instruments contain an embedded scripting engine that's capable of executing both instrument control commands and basic programming functionality.

Each TSP-enabled instrument has a set of commands that covers the entire range of functionality for that instrument. In addition to operating like a traditional command set, TSP is also a programming language. This offers two important benefits for instrument control:

- The commands themselves mimic high-level languages like C# or Python.
- The instruments can execute simple programming structures such as for loops and while loops outside of the instrument specific command set.

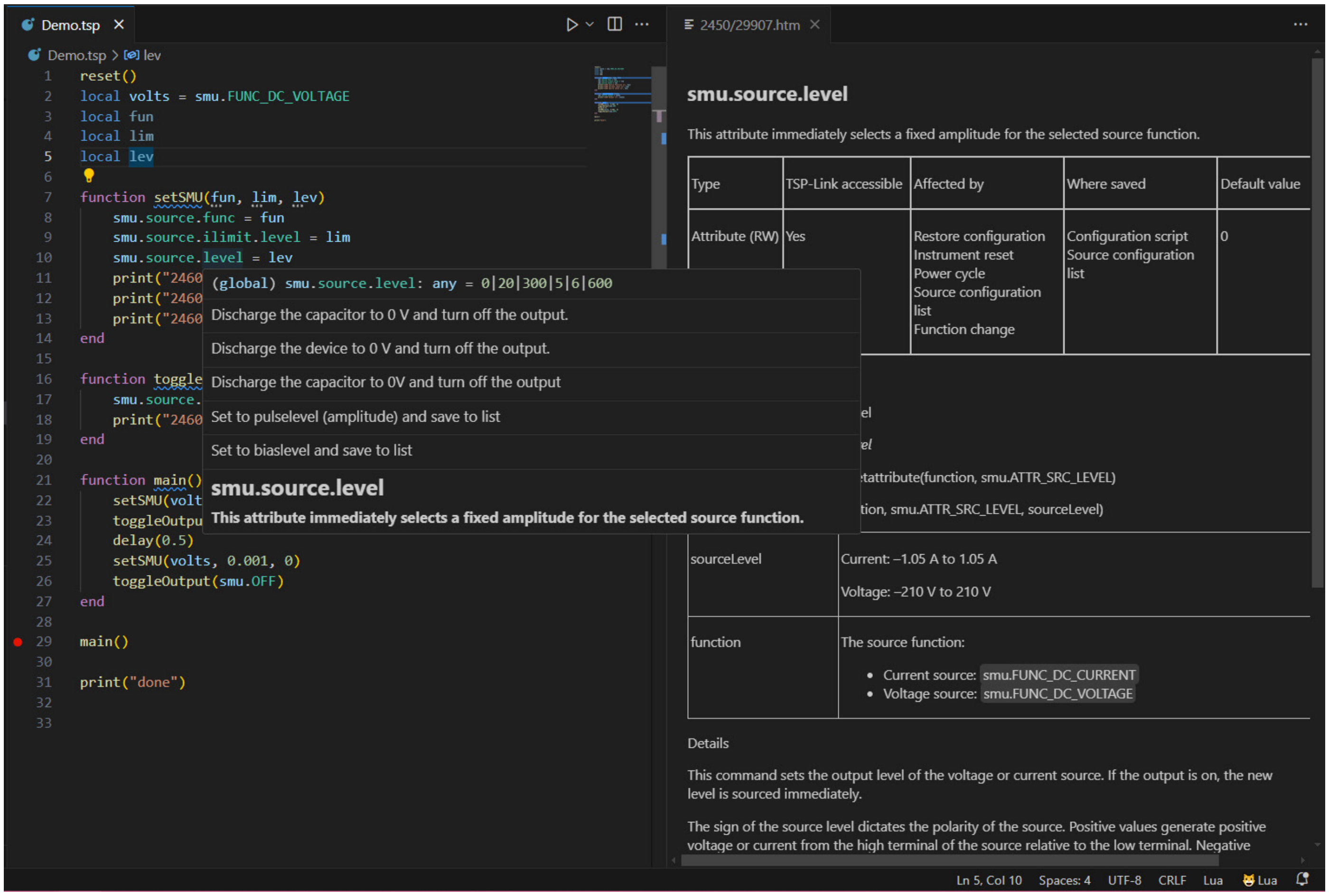
With TSP for test automation, we can create a file called a TSP script that contains all or part of the test routine. The TSP script can execute sweeps and collect data without receiving commands remotely or requiring front panel configuration, because it can be run directly on the instrument. Unlike instruments that are only compatible with SCPI, TSP-enabled instruments can store full scripts, functions or variables in their memory, allowing it to function autonomously once loaded with a script. For writing these scripts, TSP Toolkit is the obvious choice.

TSP Toolkit

TSP Toolkit is an updated script development environment taking the form of a Microsoft Visual Studio Code extension that includes instrument support for Keithley TSP-enabled instrumentation such as source measure units (SMUs), digital multimeters (DMMs) and data acquisition systems (DAQs), plus many quality-of-life features that improve the script development experience. TSP Toolkit is replacing Keithley Test Script Builder (TSB) and features all the same functionality plus more.

TSP Toolkit features the modern user interface (UI) of the Microsoft Visual Studio Code editor (VS Code), complete with syntax-highlighting for TSP to increase readability. VS Code also offers hundreds of additional extensions, so users developing with multiple programming languages can seamlessly integrate TSP script development into their workflows.

The TSP Toolkit extension also features TSP command autocompletion alongside in-line and hover help, eliminating the need to manually parse through dense reference manuals to confirm proper command usage and syntax.



```
reset()
local volts = smu.FUNC_DC_VOLTAGE
local fun
local lim
local lev

function setSMU(fun, lim, lev)
    smu.source.func = fun
    smu.source.ilimit.level = lim
    smu.source.level = lev
    print("2460
    print("2460
    print("2460
end

function toggle
    smu.source.
    print("2460
end

function main()
    setSMU(volt
    toggleOutpu
    delay(0.5)
    setSMU(volts, 0.001, 0)
    toggleOutput(smu.OFF)
end

main()

print("done")
```

```
(global) smu.source.level: any = 0|20|300|5|6|600
```

Discharge the capacitor to 0 V and turn off the output.

Discharge the device to 0 V and turn off the output.

Discharge the capacitor to 0V and turn off the output

Set to pulselevel (amplitude) and save to list

Set to biaslevel and save to list

smu.source.level

This attribute immediately selects a fixed amplitude for the selected source function.

smu.source.level

This attribute immediately selects a fixed amplitude for the selected source function.

Type	TSP-Link accessible	Affected by	Where saved	Default value
Attribute (RW)	Yes	Restore configuration Instrument reset Power cycle Source configuration list Function change	Configuration script Source configuration list	0

el

el

tattribute(function, smu.ATTR_SRC_LEVEL)

tion, smu.ATTR_SRC_LEVEL, sourceLevel)

sourceLevel	Current: –1.05 A to 1.05 A Voltage: –210 V to 210 V
function	The source function: • Current source: smu.FUNC_DC_CURRENT • Voltage source: smu.FUNC_DC_VOLTAGE

Details

This command sets the output level of the voltage or current source. If the output is on, the new level is sourced immediately.

The sign of the source level 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

Figure 1: TSP Toolkit command hover help feature in use.

TSP Toolkit can easily connect to TSP-enabled instruments via the instrument pane, which comes equipped with instrument autodiscovery. Discovered instruments in the instrument pane can be expanded to view their model, serial number, VISA or IP address and port number.

Right-click on a discovered instrument to access firmware upgrades, the ability to rename your instrument and establish a connection to the instrument.

Upon connection, an instrument terminal launches. This terminal can function the same as any command line terminal, including the one present in Test Script Builder. You can have multiple terminals open at once, giving you the option to connect to multiple instruments at a time.

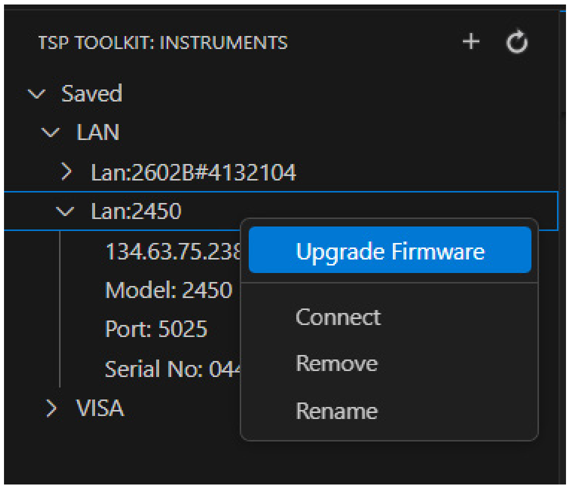
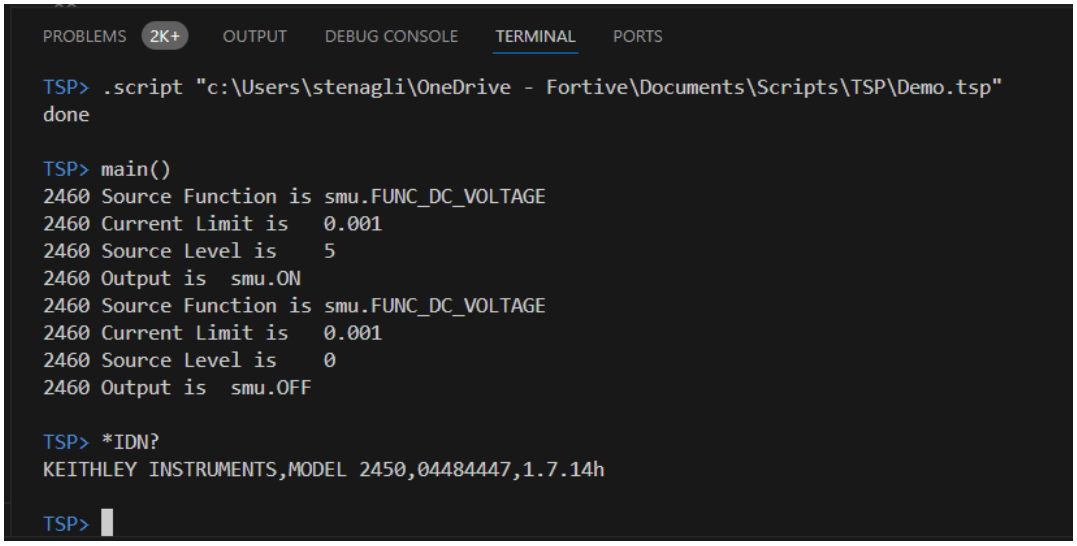


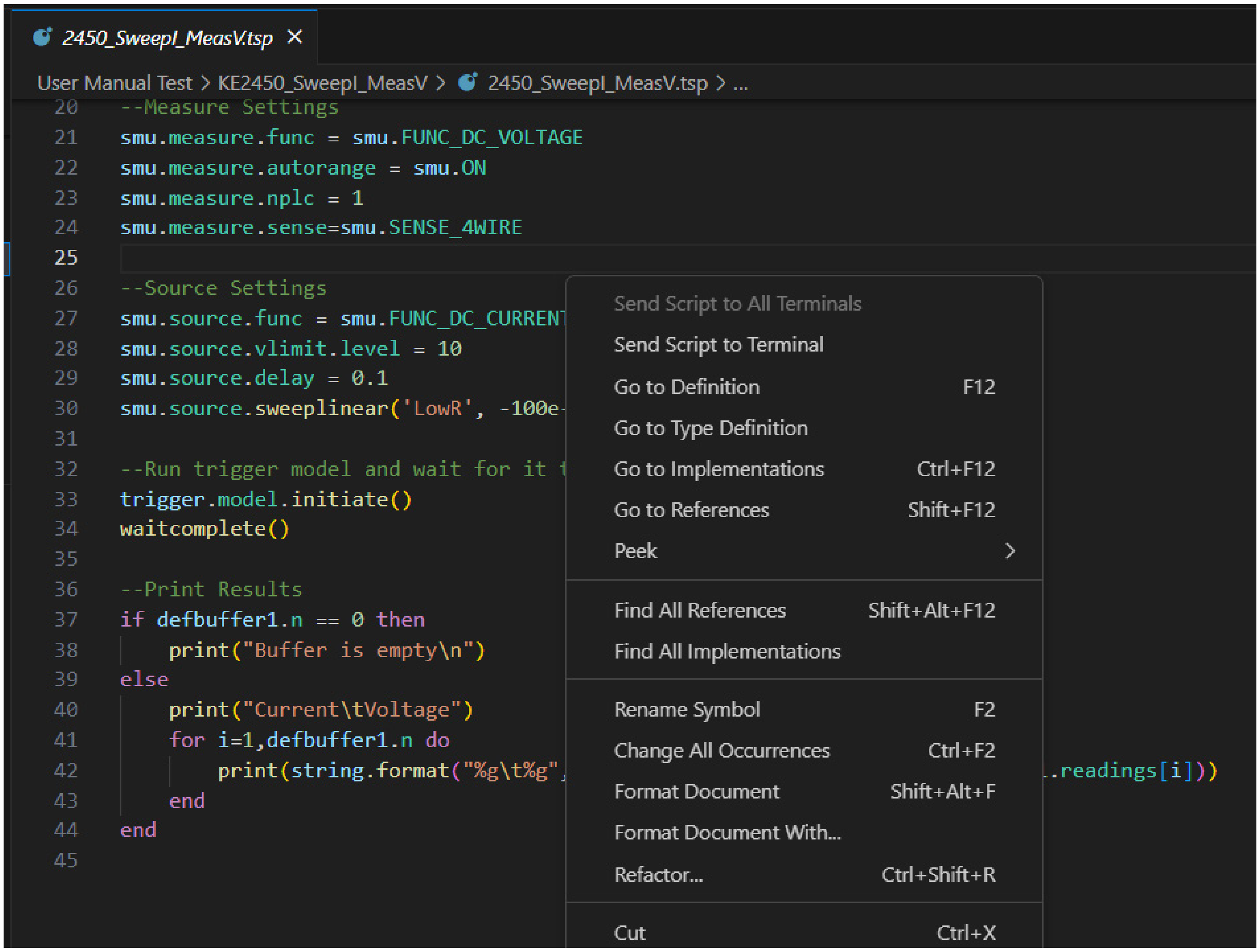
Figure 2: TSP Toolkit instrument pane context menu options.



```
TSP> .script "c:\Users\stenagli\OneDrive - Fortive\Documents\Scripts\TSP\Demo.tsp"
done

TSP> main()
2460 Source Function is smu.FUNC_DC_VOLTAGE
2460 Current Limit is   0.001
2460 Source Level is    5
2460 Output is  smu.ON
2460 Source Function is smu.FUNC_DC_VOLTAGE
2460 Current Limit is   0.001
2460 Source Level is    0
2460 Output is  smu.OFF

TSP> *IDN?
KEITHLEY INSTRUMENTS,MODEL 2450,04484447,1.7.14h

TSP>
```

Figure 3: TSP Toolkit instrument terminal in use.

The terminal can be used to send individual TSP commands to the instrument or even execute function calls from pre-loaded scripts. The terminal is also where any errors or queries are returned.


2450_SweepI_MeasV.tsp
User Manual Test > KE2450_SweepI_MeasV > 2450_SweepI_MeasV.tsp > ...
Send Script to All Terminals
Send Script to Terminal
Go to Definition F12
Go to Type Definition
Go to Implementations Ctrl+F12
Go to References Shift+F12
Peek
Find All References Shift+Alt+F12
Find All Implementations
Rename Symbol F2
Change All Occurrences Ctrl+F2
Format Document Shift+Alt+F
Format Document With...
Refactor... Ctrl+Shift+R
Cut Ctrl+X

```
--Measure Settings
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
smu.measure.sense=smu.SENSE_4WIRE

--Source Settings
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.vlimit.level = 10
smu.source.delay = 0.1
smu.source.sweeplinear('LowR', -100e-

--Run trigger model and wait for it t
trigger.model.initiate()
waitcomplete()

--Print Results
if defbuffer1.n == 0 then
    print("Buffer is empty\n")
else
    print("Current\tVoltage")
    for i=1,defbuffer1.n do
        print(string.format("%g\t%g",                                          ..readings[i]))
    end
end
```

Figure 4: TSP Toolkit script editor window context menu options.

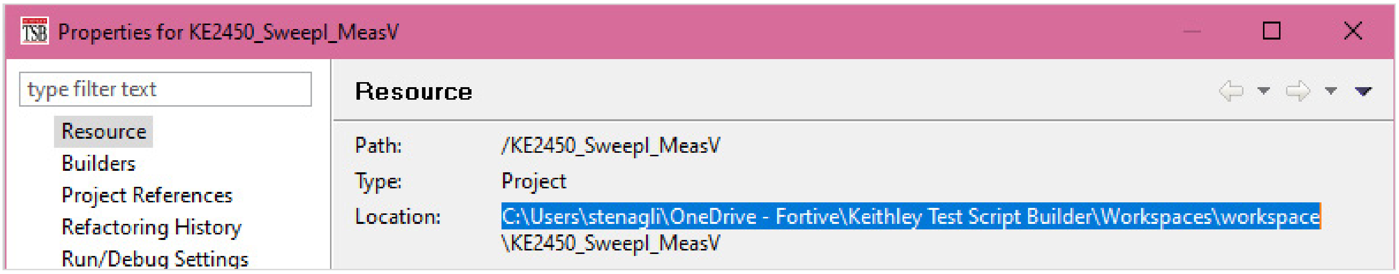
To run your script, right click anywhere within the script editor window and select “Send Script to Terminal”, or, if you have multiple connections and want the script to run on all connected instruments, select “Send Script to All Terminals.”

The script will be sent to the instrument and run directly on the box, which improves throughput by reducing the number of interactions across the bus and takes the processing burden away from the PC.

Saving Example Scripts from Test Script Builder

If you are a user of the original Keithley Test Script Builder, you know that it includes a library of example TSP scripts. TSP Toolkit enables easy migration of these scripts and those developed with TSB.

To use TSB examples in TSP Toolkit, find the TSB Workspace file on your PC and copy it to a local directory on your PC. You can find where the TSB Workspace file is stored on your computer by right clicking on any of the examples in the Navigator tab and selecting properties.


Properties for KE2450_SweepI_MeasV

type filter text

Resource
Builders
Project References
Refactoring History
Run/Debug Settings

Resource

Path: /KE2450_SweepI_MeasV
Type: Project
Location: C:\Users\stenagli\OneDrive - Fortive\Keithley Test Script Builder\Workspaces\workspace \KE2450_SweepI_MeasV

Figure 5: Keithley Test Script Builder example workspace file location.

Open this new directory in VS Code by clicking File -> Open Folder. This will give you access to all the files in the target directory via the VS Code Explorer tab.

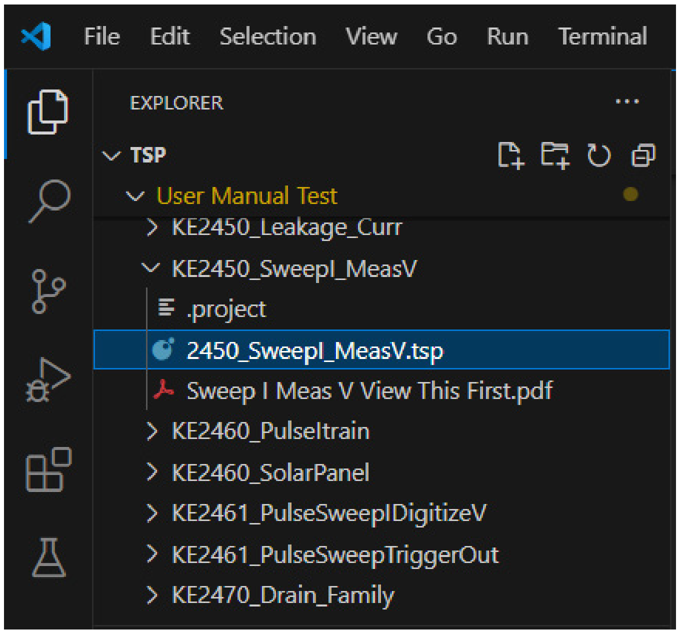


Figure 6: Microsoft Visual Studio Code explorer tab.

Example scripts are also available for download on the official Keithley GitHub.

TSP Scripting Tips & Tricks

Scripting Rules

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.

1. Each script must have a unique name that cannot start with a number.
2. Script names must not contain spaces.
3. Script names must be fewer than 27 characters long.
4. 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.
5. 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.
6. 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.

It should be noted that these rules mainly apply to scripts that are transferred to an instrument's memory via USB or another scripting language like Python. This is because when running scripts within TSP Toolkit, the extension takes care of many of these rules for you — but the script name character limit must still be observed.

Aliasing

If you do not like the naming conventions of traditional TSP commands or wish to abbreviate them, you can rename the commands via aliasing. TSP commands are structured in levels, separated by periods. A portion of these levels or the entire command can be stored to a variable and then the variable is used to call the command. With every level aliased, the script performance will be better, and the commands are processed faster.

This example demonstrates how to create aliases for TSP commands:

TSP Toolkit Example 1: Aliasing

```
local dm = dmm.measure
local current = dmm.FUNC_DC_CURRENT
dm.func = current -- Alias for dmm.measure.func = dmm.FUNC_DC_CURRENT

local stop = trigger.model.abort
stop() -- Alias for trigger.model.abort()

local clear = eventlog.clear
clear() -- Alias for eventlog.clear()
```

Note in the above example, that the first line is aliasing 2 levels of a command, the second line is aliasing an enum and the third line is using the aliases together to set the measure function to current.

When aliasing a function command like trigger.model.abort(), the parenthesis normally accompanied with the function is not used. However, when calling the aliased function, stop(), the parenthesis is used normally. This behavior is the same for any command that is a function.

Store & Run Scripts Locally

TSP scripts can also be saved to the instrument's internal memory. This allows them to be accessed by any remote control scheme or even be run without a controlling PC from the instrument's front panel.

Scripts may be sent and saved to an instrument by using the loadscript and endscript keywords. This is especially helpful when you need to automate the delivery of scripts, or when it is more practical to send scripts over an existing remote connection. Note that the use of these keywords is not required for running the script in TSP Toolkit. They are used only if you want to send the script to the instrument's memory without executing it so that it can be saved on the instrument itself.

TSP Toolkit Example 2: Using the Loadscript and Endscript Keywords

```
loadscript testinfo --Send the loadscript command with a script name. This tells the instrument to
start collecting messages for a script named testInfo

display.settext(display.TEXT1, "Batch 233")
display.settext(display.TEXT2, "Test Information")
display.changescreen(display.SCREEN_USER_SWIPE)

endscript --Send the command that tells the instrument that the script is complete

testInfo() --Run the script by sending the script name followed by ()

testInfo.save() --To save the script to nonvolatile memory, send the command scriptname.save()
```

If using TSP Toolkit, a script can be saved to the instrument without the use of the loadscript and endscript keywords. Simply open a connection to the target instrument and enter:

```
.script "path/to/scriptname.tsp" -save
```

in the instrument terminal.

Another method, which is more suited for small installation bases, involves simply saving the TSP script on a USB drive. The front panel controls of TSP-enabled instruments will allow you to run the script directly from the USB drive or save the script to the instrument's internal memory. If a script on a USB flash drive is named autoinstall.tsp, the script is automatically copied to the list of internal scripts when the drive is inserted into the instrument.

Automatic Execution Scripts

Scripts saved to USB drives or the instrument's internal memory can be copied to start up. Scripts added to start up will automatically execute as part of the instrument's power on sequence. Below is an example script that changes the buffer size and sets a Keithley DMM6500 6½-Digit Bench/System Digital Multimeter to measure current:

TSP Toolkit Example 3: Measuring Current When DMM is Turned On

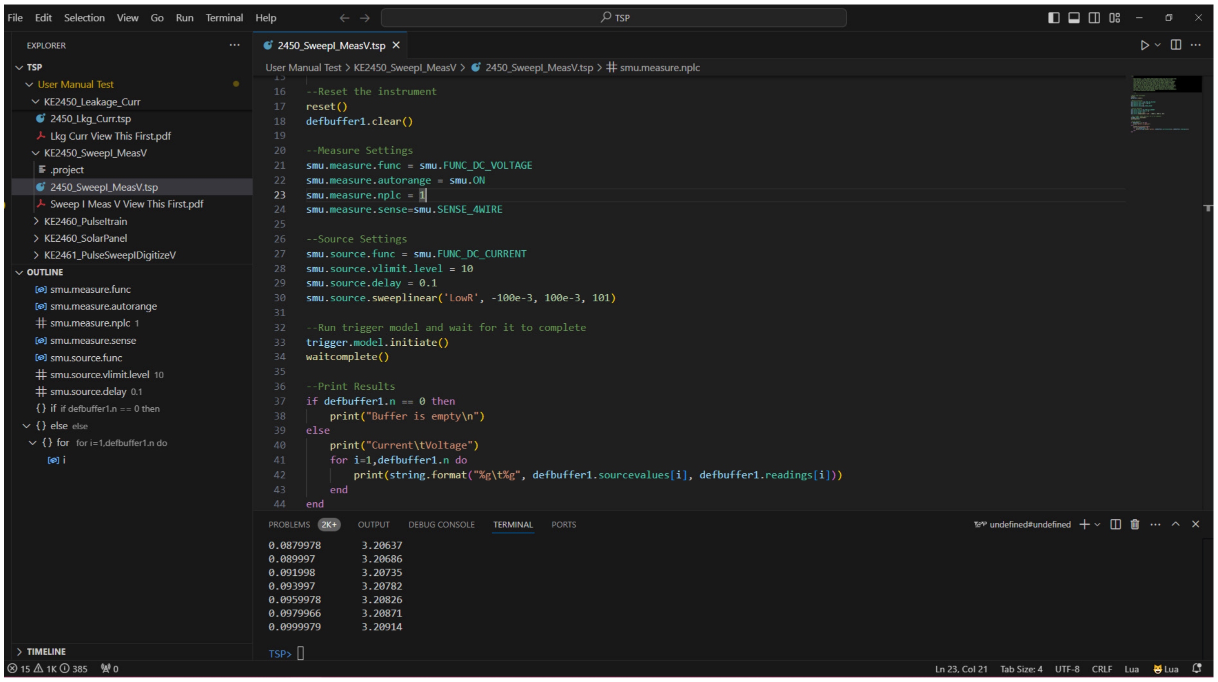
```
reset() --Reset the instrument to a known state
defbuffer1.capacity = 1000 -- Change the size of the default buffer
dmm.measure.func = dmm.FUNC_DC_CURRENT -- Set the measurement function to current
dmm.measure.read() -- Take a reading
```

To save the script to the instrument's power up sequence from TSP Toolkit, you can name your script autoexec.tsp and the instrument will always run the script at startup.

Application Example: Sweeping with a Keithley 2450 SourceMeter® SMU Instrument

As mentioned previously, scripts opened or developed in TSP Toolkit can be sent to the terminal to be run on a connected instrument. In **Figure 7**, an example script from Test Script Builder has been opened within TSP Toolkit and executed. This example, 2450_SweepI_MeasV.tsp, can be downloaded from the Keithley GitHub and instructs the 2450 SMU to output a sweep from -100 mA to 100 mA in 101 steps while measuring the resulting voltage drop across the device under test.

The resulting current and voltage readings are printed to the terminal. You can copy and paste these readings into a spreadsheet for further analysis and graphing.



```
--Reset the instrument
reset()
defbuffer1.clear()

--Measure Settings
smu.measure.func = smu.FUNC_DC_VOLTAGE
smu.measure.autorange = smu.ON
smu.measure.nplc = 1
smu.measure.sense=smu.SENSE_4WIRE

--Source Settings
smu.source.func = smu.FUNC_DC_CURRENT
smu.source.vlimit.level = 10
smu.source.delay = 0.1
smu.source.sweeplinear('LowR', -100e-3, 100e-3, 101)

--Run trigger model and wait for it to complete
trigger.model.initiate()
waitcomplete()

--Print Results
if defbuffer1.n == 0 then
    print("Buffer is empty\n")
else
    print("Current\tVoltage")
    for i=1,defbuffer1.n do
        print(string.format("%g\t%g", defbuffer1.sourcevalues[i], defbuffer1.readings[i]))
    end
end
```

```
0.0879978       3.20637
0.089997        3.20686
0.091998        3.20735
0.093997        3.20782
0.0959978       3.20826
0.0979966       3.20871
0.0999979       3.20914

TSP>
```

Figure 7: 2450_SweepI_MeasV.tsp example script running in TSP Toolkit.

For the purposes of this application note, this test was run on an LED with a 20 mΩ resistance. The following image is a screenshot from the 2450 front panel after the script finished executing, showing the collected data in a graph.

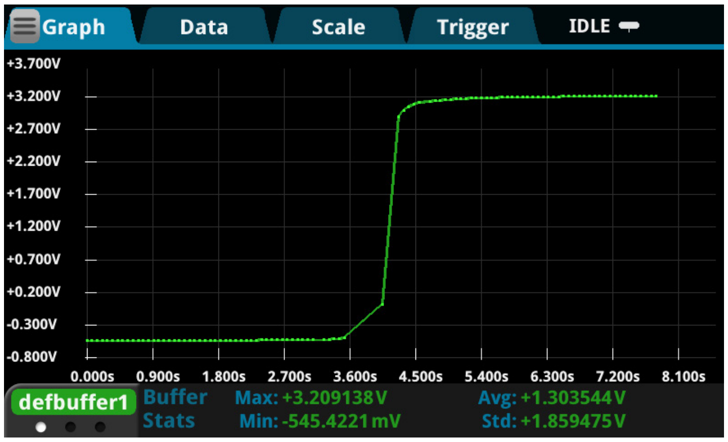


Figure 8: Resulting data shown on 2450 SMU front panel graph view.

Due to the highly extensible nature of Visual Studio Code, TSP Toolkit exists among other extensions for other programming languages. This means that you can use TSP Toolkit to write a TSP script and then run the TSP file in your Python or other framework.

The following example is a Python script that connects to an instrument via a VISA driver and then loads the 2450_SweepI_MeasV.tsp script onto the instrument using a for loop. Once the script has been loaded into the instrument's memory, it can be executed on the instrument from Python using the *.run() function.

TSP Toolkit Example 3: Measuring Current When DMM is Turned On

```
import pyvisa as visa
rm = visa.ResourceManager()
smu_address = "TCPIP0::134.63.75.238::inst0::INSTR"
smu = rm.open_resource(smu_address)
smu.timeout = 20000

file_path = "C:\\Users\\stenagli\\OneDrive - Fortive\\Documents\\Scripts\\TSP\\User Manual
Test\\KE2450_SweepI_MeasV\\"
file_name = "2450_SweepI_MeasV.tsp"
file_path_and_name = file_path + file_name
print(file_path_and_name)

smu.write("loadscript SweepI_MeasV")
with open(file_path_and_name) as fp:
    for line in fp:
        smu.write(line)

smu.write("endscript")
smu.write("SweepI_MeasV.run()")
smu.close()
```

Conclusion

Using TSP as a powerful scripting tool can increase the overall functionality of your instruments and increase productivity by creating and running test scripts faster. Scripting gives users the option to control multiple instruments with a single program and significantly reduces communications over the bus. Fully integrating logical operations with remote commands creates a plethora of possibilities for new tests, and TSP Toolkit makes it easier than ever to get started. Visit the TSP Toolkit product page to learn more and download.

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

KEITHLEY
A Tektronix Company

Find more valuable resources at TEK.COM