KEITHLEY

Application Note Series

Number 2875

Converting a Series 2700 SCPI Application to a Series 3700 System Switch/ Multimeter System Script Application

Introduction

For many years, instrument manufacturers have used "Standard Commands for Programmable Instrumentation" or SCPI to control programmable test and measurement devices in instrumentation systems. SCPI provides a uniform and consistent language for the control of test and measurement instruments. The same commands and responses control corresponding instrument functions in SCPI equipment, regardless of the manufacturer or the instrument type.

By design, Keithley's new Series 3700 System Switch/ Multimeter instrument does not use SCPI commands. The platform instead uses an internal Test Script Processor (TSP) to process and run programs called "scripts." The migration to TSP was to address the more demanding system throughput requirements common to today's systems. Communication to any TSP instrument is the same as interfacing to a conventional instrument using commercially available software applications and application development environment. The user simply sends text strings containing either TSL or ICL commands to the instrument using the communication interface. Although Keithley provides an application suite called Test Script Builder (TSB), it is not intended as the only tool to communicate to the instrument, nor does not replace commercially available software applications. TSB is a basic tool available to use to develop scripts for TSP instruments.

A script is a collection of instrument control commands and/ or program statements. Program statements control script execution and provide facilities such as variables, functions, branching, and loop control. TSB is a programming interface in which users can create powerful, high speed, multi-channel tests to download into either the unit's volatile or nonvolatile memory. Using the downloaded script, the unit controls itself, independent of the system's host controller. This capability can free up the system controller to interface with other equipment in the rack more frequently, thereby increasing the overall system throughput. Furthermore, the Series 3700 has very deep memory; program memory can hold 50,000 lines of code and data memory can store at least 450,000 readings.

This application note compares the Series 2700 SCPI-based applications with the new Series 3700 scripting approach.

Comparing SCPI to ICL Commands and Scripts

The SCPI Instrument Model

Some measurements require direct control over an instrument's hardware. To provide this control, SCPI-based instruments contain command subsystems that control particular instrument functions and settings.

The SCPI Instrument Model is divided among the SCPI command subsystems. In the case of the Series 2700, the command subsystems are broken down into these categories:

1. Signal-Oriented Measurement: Commands used to acquire readings.
2. Calculate: Used for math expressions, limit testing, and statistics.
3. Display: Controls the display of the Integra instruments.
4. Format: Selects the data format for transferring readings over the bus.
5. Route: Controls front/rear inputs or switching.
6. Sense: Configures and controls the measurement functions.
7. Status: Controls the status registers.
8. System: Contains miscellaneous commands for instrument setup.
9. Trace: Configures and controls data storage into the buffer.
10. Trigger: Configures the Trigger Model.
11. Unit: Configure the query the displayed measurement units

The TSP Instrument Model

Test Script Language (TSL) is the language of the Series 3700. ICL is a group of predefined functions and variables that are used to control the instrument. They are the instrument commands that are equivalent to SCPI commands in the SCPI instruments. The following command sets apply to the Series 3700:

1. Beeper: Commands used to control the built-in beeper.
2. Bit: Used to perform logic operations on one or two numbers.
3. Delay: Used to control read/write and trigger operations for the digital I/O port.
4. Digital I/O: Selects the data format for transferring readings over the bus.
5. Display: Used to control display messaging on the front panel of the Model 3700.
6. Error Queue: Used to read the entries in the error/event queue.
7. Exit: Used to terminate a script that is presently running.

8. Format: Used for data printed with the printnumber and printbuffer commands.
9. GPIB: Used to set the GPIB address.
10. LocalNode: Used to set the power line frequency, control (on/off) prompting, and control (hide/show) error messages on the display.
11. Make: Used to set and retrieve a value for an attribute.
12. Operation Complete: Sets the OPC bit in the status register when all overlapped commands are completed.
13. PrintBuffer: Used to print data and numbers.
14. Reset: Used to return a Model 3700 to its default settings.
15. Setup: Used to save/recall setups and set the power-on setup.
16. Trigger: Used to control triggering.
17. TSPLink: Used to assign node numbers to a mainframe and initialize the TSP-Link system.
18. UserString: Used to store/retrieve user-defined strings in non-volatile memory.
19. Wait Complete: Waits for all overlapped commands to complete.

What may look like a larger list of command definitions compared to the Series 2700 is instead a reduced set of individual commands. For example, the SCPI subsystem for "Calculate" is handled mostly through scripting; therefore, no ICL commands exist for those functions. See ***Table 1*** at the end of this document for a list of ICL commands and the corresponding SCPI commands.

There are two alternatives for programming and communicating with the Series 3700: either executing individual ICL commands (similar to sending individual SCPI commands) or writing test scripts using Test Script Language. Test Script Language (TSL) is a programming language based on Lua, a standard programming language (www.lua.org). TSL is capable of performing branching, looping, and other attributes for the purpose of controlling instruments using ICL commands.

Scripts are a collection (list) of instrument control commands (ICL) and/or program statements (TSL). Series 3700 instruments execute all commands and statements in a script. Running a script at the instrument level is faster than running the test program from the PC. Using scripts eliminates the time needed for transmission from PC to instrument via GPIB. Thanks to the Test Script Processor (TSP) built into the instrument, an entire TSL control program can be loaded in the instrument. Then, sending a single command can execute the whole program. In other circumstances, the applications might be better served if some portion of the program resides on the PC. The beauty of scripting is its ability to divide up the program in any way that makes sense.

To compare the difference between using SCPI commands and ICL commands, let's look at the two command sets in an example that performs a simple scan. The Series 2700 SCPI commands and the equivalent Series 3700 TSP script are shown in Example 1. A complete comparison of the Series 3700 TSP language commands versus the comparable SCPI commands for the Series 2700 is listed in ***Table 1***.

Example 1: Volts Measurement/Simple Scan Channels 1–20

The following example code (Example 1) will:

- Set channels for DC Volts
- Specify a scan list from channels 1 through 20
- Return the voltage readings to the host PC

Example 1.

SCPI Command	Comments
*RST	Restore GPIB defaults
:SENS:FUNC 'VOLT', (@101:120)	Configures for DCV function with channel list
:ROUT:SCAN(@101:120)	Specify channel to scan
:SAMP:COUN 20	Sample count to 20
:TRIG:COUN 1	Trigger one scan
:ROUT:SCAN:LSEL INT	Enable scan mode
:TRAC:FEED:CONT NEXT	Enable trace Buffer
INIT	Init scan
DATA?	Request buffer readings

ICL command Script	Comments
`Reset()`	Restore GPIB defaults
`dmm.setconfig("1001:1020","dcvolts")`	Specify channel to scan and associated function
`mybuffer=dmm.makebuffer(20)`	Creates a 20-reading user buffer named mybuffer
`reading=dmm.measure(mybuffer)`	
`scan.create("1001:1020")`	Creates a scan list for channels 1–20 on slot 1
`scan.scancount=1`	Trigger one scan
`scan.execute(mybuffer)`	Specifies mybuffer as the reading buffer for the scan to use
`print(printbuffer(1,20, mybuffer))`	Print contents of my buffer

The SCPI program shown in Example 1 converts easily to an equivalent ICL script. Note the similarity in structure of the ICL commands to the SCPI commands. One difference is how the readings are taken. In the SCPI protocol, getting a reading is actually a two-step process. First, it's necessary to ask for a reading using one of several query commands. This example used the "`DATA?`" command to retrieve the data from the internal buffer. Once the query is sent, the readings are stored in a reading queue. The control program must then get the reading from the queue to complete the process. If any further commands or queries are sent without getting the complete reading from the reading queue, the instrument will give a -410 query-interrupted error. These -410 query-interrupted errors are a result of interrupting the query and are a common pitfall for SCPI-based products.

In the SCPI instrument, the `INIT` command is sent to initiate the scan, and the measurements are always automatically stored into an internal buffer. This is not the case with the ICL commands. Note that in the code in Example 1, the scan is initiated by the `scan.execute(mybuffer)` command. In TSL, a buffer named `mybuffer` was created to store the measurements. Another option is to create a variable, then the variable can be used within the 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.

TSP language goes well beyond just sending instrument commands; it also includes capabilities like variable and variable typing, math operators and operations, tables and arrays, creation of user functions that are callable from scripts, precedence, logical operators, string concatenation, conditional branching, loop control, and built-in standard string and math callable libraries. These tools, which are built into the TSP language, open up the programming potential for the Series 3700 instruments, but more importantly, makes application development simpler than with SCPI programming.

Comparing Switching of the Series 2700 and Series 3700

Assigning Channels

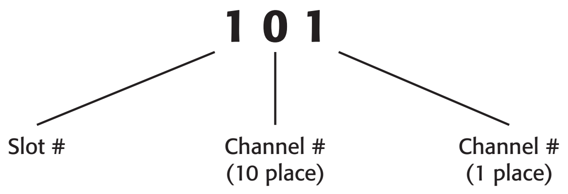
To illustrate the potential of the TSP, let's look at more SCPI Series 2700 application examples and compare them with Series 3700 scripts. But first, we'll review interpreting channel numbers. Basically, the difference between them is the Series 2700's use of three numbers versus the Series 3700's use of four. The additional number represents the one-hundredth place and is most commonly used for backplane channels.

Series 2700: (@101:110)
Series 3700 ("1031,1921")

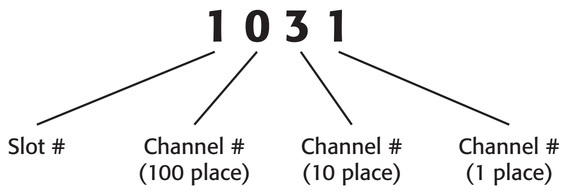
Note how both series use a comma ',' to indicate individual channels and a colon ':' to include all channels in between.

To designate channel number,

Series 2700: (@101)



Series 3700: ("1031")



Series 3700 instruments offer greater switching capabilities than the Series 2700. The channel designation includes the backplane channel in the number. Backplane channels connect the output of the switch to the measuring instrument (DMM). By default, the Series 2700 backplane relays are closed. Opening backplane channels allows disconnecting from the internal DMM. Backplane channels can be individually controlled using the ROUT:MULTI command to disconnect from the DMM input terminals. In other words, backplane channels cannot be controlled using the scan function. This limitation requires detailed management of DMM function, opening/closing the channel(s) and backplane relay(s). See the Series 2700 instrument manual for details.

By default, the Series 3700 backplane relays are open. In addition to DMM connection backplane channels, there are four other analog backplanes available to interconnect between cards. Having direct access to control the analog backplanes offers flexibility in the switching configuration not possible with the Series 2700. . The Series 3700 also offers greater control over how things are opened or closed as well as the relay connection method. As expected, with this flexibility comes multiple choices for the appropriate ICL command, as discussed in the next section.

Closing Channels

Series 2700:

The `ROUT:CLOS` command is used to close a specified channel. However, this command also executes an open all channels prior to closing the designated channel. If required to maintain channel closures while closing an additional channel, the `ROUT:MULTI` command is used.

The SCPI command `ROUT:CLOSE(@101)` performs the following sequence:

1. Breaks before make connection
2. Opens all channels previously closed
3. Closes channel 1 in slot 1 including appropriate backplane

Series 3700:

The Series 2700 offers two channel close commands (`:ROUT:CLOS` and `:ROUT:MULT:CLOS`), while the Series 3700 has four unique commands to perform a channel close. Each command executes channel close slightly differently, offering added flexibility and various advantages for choosing one over the other. For example, there are "switch only" commands that do nothing to configure the DMM, or ensure a good measurement path that provides measurement integrity. Then there are other commands that not only close the channel, but perform additional tasks to configure the DMM function to ensure the proper switches are closed for that configuration. The following is a list of the commands and what each of them do:

`dmm.close:` Automatically closes the appropriate backplane channels when a function is associated with it. By default, there is no function associated with a channel. This command will generate an error when a measurement function is not associated with the channel. The command to associate a function with a channel is the dmm.setconfig command. For example, `dmm.setconfig("1031","dcvolts")` associates the DC volts function to channel 31.

`dmm.close` performs four sequential tasks:

1. Opens channels associated with card bank
2. Opens any other backplane relay of any other cards connected to Analog Backplane 1 and 2

3. Closes the specified channel and closes the appropriate backplane relay(s) to connect the closed channel to the DMM
4. Configures the DMM as per the configuration previously assigned

`channel.close`: Similar to the `:ROUT:MULT:CLOS` command of the Series 2700, this command never changes the DMM function, open channels or close associated backplane channels (unless the backplane relay(s) are part of the channel list or configured via `channel.set.backplane` commands). Basically, this low level command strictly closes the selected channel(s). It offers the most switching and timing control because it does not perform any background or switch management tasks.

`channel.exclusiveclose`: Similar to the `ROUT:CLOS` command used in Series 2700 instruments, which closes the specified channel (and opens all others). The exclusive close opens previously closed channels and analog backplane relays on all slots and then closes desired channel(s) and specified analog backplane relays.

`Channel.exclusiveslotclose`: Opens *only* the previously closed channels in all slots associated with the channels in the channel list prior to closing the designated channel.

Backplane Channel Control

Series 2700:

By default, the Integra (Series 2700) products have a measurement function (DCV) associated with the channel. Backplane channels are automatically closed based on the measurement function selections (i.e., two-wire versus four-wire ohms).

Series 3700:

The Series 3700 differs from the Series 2700 in that it is optimized to be a stand-alone switching system. A critical difference is that there is no default function automatically associated with a channel. If a channel is not associated with a measurement function, the backplane channels are opened to allow disconnection from the internal DMM. To take a measurement, the user must associate a channel with a function to connect to the internal DMM. Then, with a close command such as the `dmm.close`, the backplane channels are automatically closed. Refer to the Series 3700 instrument manual for more details on controlling backplane channels when using these instruments strictly as a switch system.

Connection Methods

The Model 2700 offers one connection method: the break before make method. Multiple connection methods are available for the Series 3700, but connection options apply to only scan or exclusive close methods.

Series 2700:

Break before Make: This connection method breaks or disconnects the previous contact before making or connecting to the next contact. Since this is only connection schema offered for Series 2700 instruments there is no SCPI command(s) for configuration.

Series 3700:

Series 3700 offers three user-selectable connection methods:

1. Break before make: `channel.connectrule=channel.BREAK _ BEFORE _ MAKE`
   This connection method breaks or opens the contact before making or connecting to the next contact. Break before Make is the default and is the identical to the Series 2700.
2. Make before break: `channel.connectrule=channel.MAKE _ BEFORE _ BREAK`
   This connection method makes contact before opening or breaking previous contact.
3. OFF mode: `channel.connectrule=channel.OFF`
   This connection method does not guarantee connection method. It is offered to obtain the fastest execution possible. Using this method the second switch operation does not wait for the first one to complete. Although faster scan times can be achieved, it is at the cost of potentially closing more than one switch at a time. Users are advised to exercise caution when selecting this particular connection method, due to inherent risks associated with the transition of multiple relays closed at the same time (one relay is opening while another is closing in undefined sequence).

Converting a Series 2700 SCPI Application to a Series 3700 (ICL) Script

Temperature Measurement/Close Channel

Measuring temperature is a common data acquisition application. Thermocouples offer a low cost way to acquire a wide range of temperatures. To improve accuracy, a cold junction compensation (CJC) reference measurement is acquired with thermocouple measurements. Some of the Series 2700 modules, including the popular Model 7700 (20 channel) or the 7708 (40 channel) have multiple CJCs built on the module.

Series 3700 instruments implement an external screw panel accessory that houses the built-in CJC. The Model 3720 multiplexer card relay places the CJC reference on an accessory module, the Model 3720-ST terminal, which is capable of connecting up to 60 thermocouples. The 3721-ST card accessory for the 40-channel Model 3721 card also has a built-in CJC. This second example monitors a single channel for temperature measurement.

Example 2: Temperature measurement converting SCPI commands to ICL commands:

- Configures a channel for type J thermocouple measurement
- Close a channel and
- Take a reading

Model 2750 commands:

`*RST`	resets the instrument to default state
`UNIT:TEMP F`	Configure Temperature units to F
`SENS:FUNC 'TEMP'`	Configure temperature function
`SENS:TEMP:TRAN TC`	Configure Transducer for Thermocouple
`SENS:TEMP:TC:TYPE J`	Configure Thermocouple type "J"
`SENS:TEMP:TC:RJUN:RSEL INT, (@101)`	Configure Temp for internal reference
`ROUT:CLOS (@101)`	Close Channel 1
`READ?`	Initiate and request for reading

Model 3700 commands:

`reset()`	Resets to default state
`dmm.func = "temperature"`	Change function to Temperature
`dmm.transducer = dmm.TEMP_ THERMOCOUPLE`	Sets transducer type to Thermocouple
`dmm.thermocouple = dmm. THERMOCOUPLE_J`	Sets thermocouple type to "J"
`dmm.units = dmm.UNITS_ FAHRENHEIT`	Configures Temperature units to "F"
`dmm.refjunction=dmm.REF_ JUNCTION_INTERNAL`	Configures reference for internal
`dmm.configure.set ("mytemp")`	Configures for Temperature function
`dmm.setconfig("1031", "mytemp")`	Specify channel to close with associated function
`dmm.close("1031")`	Close channel 1031 in slot 1 with associated backplane channel (1921) and configures DMM
`print(dmm.measure())`	Initiates a measurement and prints the result to the output queue

Four-Wire Low Resistance Scan Example

Measuring low resistance is a typical application for the Model 2750 and Series 3700. The next example demonstrates the conversion of a Series 2700 SCPI to a Series 3700 ICL for a four-wire resistance scan. In the SCPI code, a delay period in the program is required to allow the scan time to complete prior to sending the DATA?

Example 3: Four wire Resistance compares SCPI commands to ICL commands:

- Configure 10 channels for four-wire resistance function
- Configure for enhanced accuracy: enable offset compensation, NPLC setting of 10, using the 1Ω range.
- Only the readings are returned
- Perform two scans

Model 2750 SCPI commands:

`*RST`	resets the instrument to default state
`INIT:CONT OFF`	Disable continuous initiation
`TRAC:CLE:AUTO OFF`	Clears internal 55,000 reading buffer
`FORM:ELEM READ`	Specifies Data elements to return Reading
`SENS:FUNC 'FRES',(@101:110)`	Configure channels 4W Res function
`SENS:FRES:NPLC 10, (@101:110)`	Configure Resistance for NPLC 10
`SENS:FRES:RANG 1`	Configure measurement range for 1 Ohm
`SENS:FRES:OCOM ON, (@101:110)`	Enable offset compensation feature
`TRIG:COUN 2`	Trig count for 2
`SAMP:COUN 10`	Sample count for 10
`ROUT:SCAN:LSEL INT`	Enable internal scan
`ROUT:SCAN (@101,110)`	Scan list for channels 1 & 10
`INIT`	Initiate scan
`DATA?`	Requests data stored in the buffer

These lines of SCPI commands can be directly converted to a Series 3700 TSP test sequence. The following lines of TSP code can be used to perform the same function:

Model 3700 ICL commands:

`reset()`	Resets the System Switch/Multimeter to default state
`mybuffer=dmm.makebuffer(20)`	Create 10 point buffer called mybuffer
`dmm.func="fourwireohms"`	Change function to 4-wire ohms
`dmm.nplc=10`	Configure Resistance for NPLC 10
`dmm.range=1`	Configure measurement range for 1 Ohm
`dmm.offsetcompensation=dmm.ON`	Enable offset compensation feature
`dmm.drycircuit=dmm.ON`	Enable dry circuit function
`dmm.configure.set("myfres")`	Creates a DMM configuration with attributes based on the 4 wire resistance function and associates with the name "myfres"
`dmm.setconfig("1001:1010","myfres")`	Associates "myfres" with items specified in the parameter channel list
`scan.create("1001:1010")`	Creates a new scan list using "myfres" with channels 1-10
`scan.scancount=2`	Set scan count to 2
`scan.execute(mybuffer)`	Specifies to use mybuffer during scan
`printbuffer(1,20,mybuffer)`	Print buffered readings

Note that the SCPI and ICL command structures are similar and English-like. One uses colons (SCPI) while the other uses periods (ICL), one is not case sensitive (SCPI) while the other is (ICL). But both indicate basically what the user wants to be performed. So, what is the advantage of one language over the other?

The advantage is found using the scripting compatibilities. Let's look at a script example to examine how a script can perform the same functionality as a scan. This example performs a loop in code, rather than configuring the instrument to perform a scan.

Example 4: Four-Wire Low Resistance script:

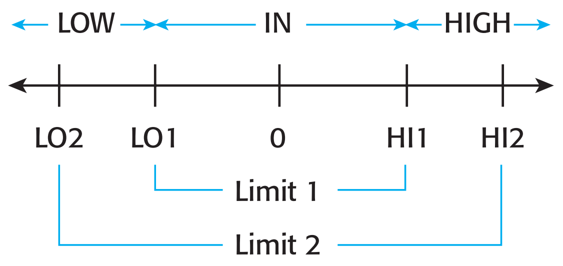
```
function 4wireR_(loops)
reset()                                  --reset
reading_buffer=dmm.makebuffer(400000)    --create reading buffer
reading_buffer.appendmode=1              --Turn Buffer append mode on
dmm.func = dmm.FOUR_WIRE_OHMS            --configure 4 wire ohms function
dmm.nplc=.1                              --configure nplc setting
dmm.range=1000                           --configure range
dmm.configure.set("myfres")              --saves configures as "myfres"
dmm.setconfig("1001:1040","myfres")      --associates configuration to channels
for z=1, loops do
for k =1, 40 do

   chan=1000+k
   chan_num=tostring(chan)
   dmm.close(chan_num)
   dmm.measure(reading_buffer)
end
end

printbuffer(1,40, reading_buffer)
channel.open("slot1")
end
```

The preceeding example is intended to demonstrate TSP script function capabilities. Instead of the instrument using an internal scan feature, scanning channels is accomplished using TSL to pass parameters that perform a simple **for… next** loop. The function `4wireR` passes the parameter (loops) while `chan=1000+k` adds "1" to the channel variable with the command dmm.close until the **for…next** loop is fulfilled.

This script function can then be loaded into the instrument and called within a program by sending "`4wireR(5)`". This would run the function and assign the value of 5 to the variable "loops." Developing a function script, in place of using a scan feature, allows for dynamic configuration of the number of loops and number of channels each time the function is run. Scripting offers advantages from a programming perspective, but using the built-in scan mode is the preferred method to squeak out a few extra milliseconds of speed. The example uses `dmm.close` to manage backplane relays as well as configure the DMM.

SCPI Series 2700 Commands to Perform Built-in Limit Testing

Both Series 2700 and Series 3700 instruments have the same built-in limits capability of two sets of limits. Both Limit 1 and Limit 2 use High and Low limits (HI1 and LO1 and HI2 and LO2). The status indication applies to the first limit that fails, even should both limits fail.



Although Limits are not a global parameter, when configured for a simple scan, the limit value and state applies to all channels in the scan. When configured as an advanced scan, each channel can have its own unique limit configuration.

The following simple command sequence configures the Model 2750 to perform a Limit 1 test on a DCV reading (default parameters). If the 100mV limit is reached, digital output #2 will be pulled low. If the –100mV limit is reached, digital output #1 will be pulled low.

`*RST`	One shot measurement mode DCV
`:CALC3:LIM1:UPP 0.1`	Set HI1 limit to 100mV
`:CALC3:LIM1:LOW -0.1`	Set LO1 limit to –100mV
`:CALC3:LIM1:STAT ON`	Enable limit 1
`:CALC3:OUTP:LSENS ALOW`	Set logic sense to active low
`:CALC3:OUTP ON`	Enable Digital outputs
`READ?`	Trigger and request reading
`:CALC3:LIM1:FAIL?`	Request result of limit 1 test

Series 3700 TSP Script to Perform Multiple Limit Testing:

Although the Series 3700 has the same built-in limit testing capabilities as the Series 2700, this next example is intended to demonstrate the power of scripting language. Similar to the previous for next loop example, this example creates a function script that can be loaded in the Series 3700. This example performs limit testing outside of the instrument's built in two limit capability.

Example 5: limit testing using script:

This script uses limits to sort resistors into three bins:

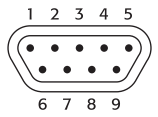
- Configures for two-wire resistance measurement
- Bins will be defined using the digital outputs
- Bin 1 is for resistors with in 1% of nominal Value
- Bin 2 is for resistors with in >1% and < 5 %
- Bin 3 is for resistors Greater than 5% of nominal Value

```
function limits_Setup_1()
nominal=100
reset()
digio.writeport(0)
dmm.func = dmm.TWO_WIRE_OHMS
dmm.nplc=1
--dmm.range=10
reading=dmm.measure()
Diff=nominal-reading
Diff=math.abs(Diff)
if Diff <= (nominal*.01) then
   print("Bin 1")
   digio.writebit(1,1)
elseif Diff > (nominal*.01) and Diff <= (nominal*.05) then-- Bin 2 (5%)
   print ("Bin 2")
   digio.writebit(2,1)
elseif Diff > (nominal*.05) then
(>5%)
   print ("Bin 3")
   digio.writebit(3,1)
end
print (reading)
print (Diff)

end
```

DIO Differences

In regard to the DIO, one difference between the two series of instruments is the Series 3700 can write directly to the DIO. Its DIO port is no longer confined to limits testing or installing a Model 7707 40-channel single-pole control module as in the Series 2700. Another obvious difference is the significant increase in available digital I/O lines. Series 3700 offers 14 digital I/O lines compared to only four lines for the Series 2700.

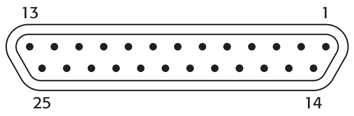
For the Series 2700, either a beeper or the DIO output line is controlled via limit testing. There are five digital output lines (pins 1–5) controlled by 2 sets of limits. The digital I/O interface is a DB-9 connector on the rear panel. ***Figure 3*** illustrates the pin-out configuration.



1 = Digital Output #1 (Low Limit 1 or LO1)
2 = Digital Output #2 (High Limit 1 or HI1)
3 = Digital Output #3 (Low Limit 2 or LO2)
4 = Digital Output #4 (High Limit 2 or HI2)
5 = Digital Output #5 (Master Limit)
6 = Ext. Trigger (Input)
7 = Diode Clamp
8 = Hardware Interlock (Input)
9 = Digital (Chassis) Ground

Figure 3

The 370X's digital I/O interface is a DB-25 connector on the rear panel. ***Figure 4*** illustrates the pin-out configuration.



1 = Digital I/O #1
2 = Digital I/O #2
3 = Digital I/O #3
4 = Digital I/O #4
5 = Digital I/O #5
6 = Digital I/O #6
7 = Digital I/O #7
8 = Digital I/O #8
9 = Digital I/O #9
10 = Digital I/O #10
11 = Digital I/O #11
12 = Digital I/O #12
13 = Digital I/O #13
14 = Digital I/O #14
15–21 = Ground
22 = +5V
23 = +5V
24 = Output Enable
25 = +5V

Figure 4

Conclusion

It is well recognized that communications between test equipment and the system controller can be a significant bottleneck that limits test system throughput. It is common practice to perform command and data transfers while a prober or handler is performing mechanical operations to avoid consuming valuable test time. However, as test systems have become increasingly complex and more controller/instrument interaction is generally required, this is increasingly difficult to achieve. Keithley's Series 3700 instruments address this throughput issue with the Test Script Processor (TSP). Loading the script on the instrument as a TSP function will minimize communication requirements. Once loaded in the Series 3700 instrument, the function can be called and reused at any time.

This application note touched on just a few of the potential applications that can be converted from a Series 2700 SCPI program to a Series 3700 TSP language script. ***Table 1*** maps Series 3700 TSP language commands to their equivalent Series 2700 SCPI commands. The Series 3700 commands have been optimized for greater flexibility of operation than to the Series 2700 SCPI commands. In many cases, there are no equivalent SCPI commands. The table simplifies reviewing existing Series 2700 SCPI programs to determine if there is an equivalent Series 3700 command. Once it's been determined that a command maps, the power of the Series 3700 TSP language allows creating reusable test sequence scripts to address many challenging test applications.

Table 1. Series 3700 ICL/2700 SCPI Command Comparison

TSP Command	Series 2700 Command(s)
Beeper Commands	
beeper.enable = 0/1	SYSTem:BEEPer:STATe
beeper.beep(duration,frequency)	N/A
Bit Commands	
bit.bitand(value1,value2)	N/A
bit.bitor(value1,value1)	N/A
bit.bitxor(value1,value2)	N/A
bit.clear(value1,index)	N/A
bit.get(value1,index)	N/A
bit.getfield(value1,index,width)	N/A
bit.set(value1,index)	N/A
bit.setfield(value1,index,width,fieldvalue)	N/A
bit.test(value1.index)	N/A
bit.toggle(value1,index)	N/A
Channel Commands	
channel.close(ch list)	ROUTe:MULTiple:CLOSe
channel.connectrule = rule	N/A
channel.connectsequential = channel.ON	N/A
channel.exclusiveclose(ch list)	N/A
channel.exclusiveslotclose(ch list)	NA
channel.getbackplane(ch _ list)	N/A
channel.getclose(ch _ list)	ROUTe:CLOSe?
channel.getcount(ch _ list)	N/A
channel.getdelay(ch _ list)	N/A
channel.getimage(ch _ list)	N/A
channel.getlabel(ch _ list)	N/A
channel.getpole(ch _ list)	N/A
channel.getstate(ch _ list)	ROUTe:CLOSe:STATe? Query closed channels in specified list :ROUTe:MULTiple:CLOSe:STATe? Query closed channels in specified list
channel.open(ch _ list)	:ROUTe:OPEN:ALL
channel.pattern.catalog()	N/A
channel.pattern.delete(name)	N/A
channel.pattern.getimage(name)	N/A
channel.pattern.setimage(ch _ list,name)	N/A
channel.pattern.snapshot(name)	N/A
channel.reset(ch _ list)	N/A
channel.setbackplane(ch _ list,abuslist)	N/A
channel.setdelay(ch _ list,value)	N/A
channel.setlabel(ch _ list,label)	N/A
channel.setpole(ch _ list,value)	N/A
Delay Command	
Delay *	TRIGger:DELay *
Digital I/O Commands	
digio.readbit(N)	OUTPut:TTL:LSENse?
digio.readport()	OUTPut:TTL:LSENse?
digio.trigger[N].assert()	N/A
digio.trigger[N].clear()	CALCulate3:LIMX:CLEar[IMMediate} @
digio.trigger[N].mode	CALCulate3:MLIMit:LATched @ :CALCulate3:OUTPut[:STATe :CALCulate3:OUTPut:LSENse <AHIGh or ALOW> :CALCulate3:OUTPut:PULSE[:STATe]
digio.trigger[N].overrun	N/A
digio.trigger[N].puslewidth	CALCulate3:OUTPut:PULSe:TIME
digio.trigger[N].release()	N/A
digio.trigger[N].stimulus	N/A
digio.trigger[N].wait (timeout)	N/A
digio.writebit	N/A
digio.writeport	N/A
digio.writeprotect	N/A
Display Command	
display.clear()	DISPlay:WINDow1:TEXT:DATA
display.getannunciators()	N/A
display.getcursor()	N/A
display.getlastkey()	N/A
display.gettext()	DISPlay:WINDow1:TEXT:DATA
display.inputvalue(format)	N/A
display.loadmenu.add(displayname,chunk)	N/A
display.loadmenu.delete(displayname)	N/A
display.locallockout	N/A
display.menu	N/A
display.prompt(format, units, help)	N/A
display.screen	DISPlay:WINDow1:TEXT:STATe
display.sendkey(keycode)	SYSTem:KEY
display.setcursor(row,column)	N/A
display.settext("text")	DISPlay:WINDow1:TEXT:DATA

TSP Command	Series 2700 Command(s)
DISPlay:WINDow1:TEXT:STATe	
display.waitkey()	N/A
DMM FUNCTIONS	
dmm.adjustment.count	N/A
dmm.adjustment.date	N/A
dmm.aperture	:SENSe:VOLTage:APERture <n> :SENSe:CURRent:APERture <n> :SENSe:VOLTage:AC:APERture <n> :SENSe:CURRent:AC:APERture <n> :SENSe:RESistance:APERture <n> :SENSe:FRESistance:APERture <n> :SENSe:TEMPerature:APERture <n> :SENSe:FREQuency:APERture <n> :SENSe:PERiod:APERture <n>
dmm.autodelay	N/A
dmm.autorange	:SENSe:VOLTage:RANGe:AUTO <n> :SENSe:CURRent: RANGe:AUTO <n> :SENSe:VOLTage: RANGe:AUTO <n> :SENSe:CURRent: RANGe:AUTO <n> :SENSe:RESistance: RANGe:AUTO <n> :SENSe:FRESistance: RANGe:AUTO <n>
dmm.autozero	:SYSTem:AZERo <n>
dmm.calibration.ac	CAL:AC:STEPx
dmm.calibration.dc	CALDC:STEPx
dmm.calibration.lock	CAL:PROT:LOCK
dmm.calibration.password	CAL:PROT:CODE
dmm.calibration.save	CAL:PROT:SAVE
dmm.calibration.unlock	CAL:PROT:CODE and CAL:PROT:INIT
dmm.calibration.verifydate	CAL:PROT:DATE
dmm.close	ROUTe:CLOSE
dmm.configure.catalog	N/A
dmm.configure.delete	N/A
dmm.configure.query	N/A
dmm.configure.recall	CONFigure
dmm.configure.set	:SENSe:FUNCtion ‘VOLTage’, <ch list> :SENSe:FUNCtion ‘CURRent’, <ch list> :SENSe:FUNCtion ‘VOLTage:AC’, <ch list> :SENSe:FUNCtion ‘CURRent:AC’, <ch list> :SENSe:FUNCtion ‘RESistance’, <ch list> :SENSe:FUNCtion ‘FRESistance’, <ch list> :SENSe:FUNCtion ‘TEMPerature’, <ch list> :SENSe:FUNCtion ‘FREQuency’, <ch list> :SENSe:FUNCtion ‘PERiod’, <ch list> :SENSe:FUNCtion ‘CONTinuity’, <ch list>
dmm.connect	N/A
dmm.dbreference	:UNITs:VOLTage[:DC]:DB:REFerence :UNITs:VOLTage:AC:DB:REFerence
dmm.detectorbandwidth	:SENSe:VOLTage:AC:DETector:BANDwidth <n> :SENSe:CURRent:AC:DETector:BANDwidth <n>
dmm.drycircuit	SENSe:FRESistance:DCIRcuit (2750 only)
dmm.filter.count	:SENSe:VOLTage[:DC]:AVERage:COUNt <n> :SENSe:CURRent[DC]:AVERage:COUNt <n> :SENSe:VOLTage:AC:AVERage:COUNt <n> :SENSe:CURRent:AC: AVERage:COUNt <n> :SENSe:RESistance: AVERage:COUNt <n> :SENSe:FRESistance:AVERage:COUNt <n> :SENSe:TEMPerature:AVERage:COUNt <n>
dmm.filter.enable	:SENSe:VOLTage[:DC]:AVERage[:STATe]<n> :SENSe:CURRent[DC]:AVERage[:STATe] <n> :SENSe:VOLTage:AC:AVERage[:STATe] <n> :SENSe:CURRent:AC: AVERage[:STATe] <n> :SENSe:RESistance: AVERage[:STATe] <n> :SENSe:FRESistance:AVERage[:STATe] <n> :SENSe:TEMPerature:AVERage[:STATe] <n>
dmm.filter.type	:SENSe:VOLTage[:DC]:AVERage:TCONtrol <name> :SENSe:CURRent[DC]:AVERage:TCONtrol <name> :SENSe:VOLTage:AC:AVERage:TCONtrol <name> :SENSe:CURRent:AC: AVERage:TCONtrol <name> :SENSe:RESistance: AVERage:TCONtrol <name> :SENSe:FRESistance:AVERage:TCONtrol <name> :SENSe:TEMPerature:AVERage:TCONtrol <name>
dmm.filter.window	:SENSe:VOLTage[:DC]:AVERage:WINDow <n> :SENSe:CURRent[DC]:AVERage:WINDow <n> :SENSe:VOLTage:AC:AVERage:WINDow <n> :SENSe:CURRent:AC: AVERage:WINDow <n> :SENSe:RESistance: AVERage:WINDow <n> :SENSe:FRESistance:AVERage:WINDow <n> :SENSe:TEMPerature:AVERage:WINDow <n>
dmm.fourrtd	:SENSe:TEMPerature:FRTD:TYPE <name>
dmm.func	:SENSe:FUNCtion <name>
dmm.getconfig	N/A
dmm.inputdivider	:SENSe:VOLTage[:DC]:IDIVider <b>

TSP Command	Series 2700 Command(s)
dmm.limit[Y].autoclear	:CALCulate3:LIMit:CLEAR:AUTO <b>
dmm.limit[Y].enable	:CALCulate3:LIMit:STATe <b>
print(dmm.limit[Y].high.fail)	:CALCulate3:LIMX:FAIL?
dmm.limit[Y].high.value	:CALCulate3:LIMit:UPPer[:DATA]
print(dmm.limit[Y].low.fail)	:CALCulate3:LIMX:FAIL?
dmm.limit[Y].low.value	:CALCulate3:LIMit:LOWer[:DATA]
dmm.linesync	:SYSTem:LSYN[:STATe]
dmm.makebuffer	N/A
dmm.math.enable	CALC1:STATe
dmm.math.format	CALCulate{1}:FORMat <NONE,MXB,PERcent or RECiprocal
dmm.math.mxb.bfactor	CALC1:KMATh:MBFactor or MA0Factor
dmm.math.mxb.mfactor	CALC1:KMATh:MMFactor or MA1Factor
dmm.math.mxb.units	CALC1:KMATh:MUNits
dmm.math.mxb.percent	CALC1:KMATh:PERCent
dmm.measure()	FETCh? READ? MEAS[:<function>]
dmm.measure(reading _ buffer)	DATa[:LATest]?
DATa:FRESh?	
dmm.measurecount = <value>	N/A
dmm.measurewithtime	N/A
dmm.nplc	:SENSe:VOLTage[:DC]:NPLCycles :SENSe:VOLTage:AC:NPLCycles :SENSe:CURRent[:DC]:NPLCycles :SENSe:CURRent:AC:NPLCycles :SENSe:RESistance:NPLCycles :SENSe:FRESistance:NPLCycles :SENSe:TEMPerature:NPLCycles
dmm.offsetcompensation = <value>	:SENSe:FRESistance:OCOMpensated
dmm.open	N/A
dmm.opendetector	TEMP:Tcouple:ODETect
dmm.range	:SENSe:VOLTage[:DC]:RANGe[:UPPer] :SENSe:VOLTage:AC:RANGe[:UPPer] :SENSe:CURRent[:DC]:RANGe[:UPPer] :SENSe:CURRent:AC:RANGe[:UPPer] :SENSe:RESistance:RANGe[:UPPer] :SENSe:FRESistance:RANGe[:UPPer]
dmm.refjunction	:TEMP:TC:RJUNction:RSELect <SIMulate,INTernal, or EXTernal>
dmm.rel.acquire	
dmm.rel.enable	:SENSe:VOLTage[:DC]:REF:STATe :SENSe:VOLTage:AC:REF:STATe :SENSe:CURRent[:DC]:REF:STATe :SENSe:CURRent:AC:REF:STATe :SENSe:RESistance:REF:STATe :SENSe:FRESistance:REF:STATe :SENSe:TEMPerature:REF:STATe :SENSe:FREQUency:REF:STATe
dmm.rel.level	:SENSe:VOLTage[:DC]:REFerence :SENSe:VOLTage:AC:REFerence :SENSe:CURRent[:DC]:REFerence :SENSe:CURRent:AC:REFerence :SENSe:RESistance:REFerence :SENSe:FRESistance:REFerence :SENSe:TEMPerature:REFerence :SENSe:FREQUency:REFerence :SENSe:PERiod:REFerence
dmm.reset()	*RST
dmm.rtdalpha	:TEMP:FRTD:ALPha (USER type constant)
dmm.rtdbeta	:TEMP:FRTD:BETA (USER type constant)
dmm.rtddelta	:TEMP:FRTD:DELTa (USER type constant)
dmm.rtdzero	:TEMP:FRTD:RZERo (USER type constant)
dmm.savebuffer	
dmm.setconfig	:SENSe:FUNCtion with channel list parameter
dmm.simreftemperature	:TEMP:TC:RJUCN:SIMulated
dmm.thermistor	:TEMP:THERmistor[:TYPE]
dmm.thermocouple	:TEMP:TCouple[:TYPE] <J,K,T,E,R,S,B,N>
dmm.threertd	N/A
dmm.threshold	:FREQ:THReshold:VOLTage:RANGe PER:THReshold:VOLTage:RANGe CONTinuity:THReshold
dmm.transducer	TEMP:TRANsducer <TCouple, FRTD or THERmistor>
dmm.units	UNIT:VOLTage[:DC] <V or DB> UNIT:VOLTage:AC <V or DB> UNIT:TEMPerature <C,CEL,F,FAR or K>
ERROR QUEUE	
errorqueue.clear()	:SYSTem:CLEar
errorqueue.count	N/A
errorqueue.next()	SYSTem:ERRor?
Exit Function	
exit	N/A

TSP Command	**Series 2700 Command(s)**
FORMAT	
format.asciiprecision	N/A
format.byteorder	:FORMat:BORDer <name> NORMal or SWAPped
format.data	:FORMat:ELEMent <item list> READing, CHANnel, UNITsm RNUMber, TSTamp and LIMits
GPIB	
Gpib.address	N/A
TRIGGER	
For lan ones, N is 0 to 7	
lan.trigger[N].assert()	N/A
lan.trigger[N].clear()	N/A
lan.trigger[N].mode	N/A
lan.trigger[N].pseudostate	N/A
lan.trigger[N].overrun	N/A
lan.trigger[N].stimulus	TRIGger:SOURce
lan.trigger[N].protocol	N/A
lan.trigger[N].wait(timeout)	N/A
localnode attributes	
localnode.linefreq	:SYSTem:LFRequency?
localnode.model	*IDN?
localnode.prompts	N/A
localnode.reset()	*RST
localnode.revision	*IDN?
localnode.serialno	*IDN?
localnode.settime	:SYSTem:TIME
localnode.setup.poweron	SYStem:POSetup
localnode.setup.recall	SYStem:POSetup
localnode.setup.save	*SAV
localnode.showerrors	System:ERRor?
opc	*OPC
reset	*RST or SYSTem:PRESet
scan.abort	:ABORT or ROUTe:OPEN ALL
scan.add(ch _ list,dmm _ config)	:ROUTe:SCAN (list)
scan.background	N/A
scan.bypass	TRIGger:TCONtrol:DIRection SOURce
scan.create(chlist,dmm _ config)	ROUTe:SCAN list
scan.execute	:INIT
scan.list	ROUTe:SCAN?
scan.mode	ROUTE:OPEN ALL
scan.reset()	*RST
scan.scancount	ARM:LAYer1:COUNt() or ARM:LAYer2:COUNt()
scan.state	N/A
scan.stepcount	TRIGger:COUNt?
scan.trigger.arm.clear()	N/A
scan.trigger.arm.set()	N/A
scan.trigger.arm.stimulus	ARM:LAYer1:SOURce () or ARM:LAYer1:SOURce()
scan.trigger.channel.clear	N/A
scan.trigger.channel.set()	N/A
scan.trigger.channel.stimulus	TRIGger:SOURce ()
scan.trigger.clear	N/A
scan.trigger.sequence.clear	N/A
scan.trigger.sequence.set	N/A
scan.trigger.sequence.stimulus	N/A
setup functions	
setup.poweron	SYSTem:POSetup
setup.recall	*RCL
setup.save	*SAV
setup.cards	ROUTe:CONFigure:SLOTX:CTYPe?
slot[X] atributes	
slot[X].commonsideohms where X is to 6 for slot number	N/A
slot[X].connectionmethod where X is to 6 for slot number	N/A
slot[X].digio where X is to 6 for slot number	N/A
slot[X].endchannel.amps where X is to 6 for slot number	N/A
slot[X].endchannel.isolated where X is to 6 for slot number	N/A
slot[X].endchannel.voltage where X is to 6 for slot number	N/A
slot[X].idn where X is to 6 for slot number	ROUTe:CONFigure:SLOTX:CTYPe?
slot[X].interlock.override	N/A
slot[X].interlock.state	N/A
slot[X].isolated where X is to 6 for slot number	N/A
slot[X].matrix where X is to 6 for slot number	N/A
slot[X].maxsettlingtime where X is to 6 for slot number	N/A
slot[X].maxvoltage where X is to 6 for slot number	N/A
slot[X].multiplexer where X is to 6 for slot number	N/A
slot[X].poles.four where X is to 6 for slot number	ROUTe:CONFigure:SLOTX:POLE?
slot[X].poles.one where X is to 6 for slot number	ROUTe:CONFigure:SLOTX:POLE?
print(slot[X].poles.two) where X is to 6 for slot number	ROUTe:CONFigure:SLOTX:POLE?
slot[X].pseudocard=<value>) where X is to 6 for slot number	ROUTe:CONFigure:SLOTX (name)

TSP Command	Series 2700 Command(s)
slot[X].startchannel.amps where X is to 6 for slot number	N/A
slot[X].startchannel.isolated where X is to 6 for slot number	N/A
slot[X].startchannel.voltage where X is to 6 for slot number	N/A
print(slot[X].tempsensor) where X is to 6 for slot number	N/A
timer functions	
timer.measure.t	N/A
timer.reset	N/A
trigger functions	
trigger.clear	N/A
trigger.wait	N/A
trigger.blender	N/A
trigger.blender[N].clear	N/A
trigger.blender[N].orenable	N/A
trigger.blender[N].overrun	N/A
trigger.blender[N].stimulus[N]	N/A
trigger.blender[N].wait where N is 1 to 4 for triggersource and N in 1 ? for blender	N/A
trigger.timer[N].clear	N/A
trigger.timer[N].count	N/A
trigger.timer[N].delay	N/A
trigger.timer[N].overrun	N/A
trigger.timer[N].passthrough	N/A
trigger.timer[N].stimulus	N/A
trigger.timer[N].wait where N is 1 to ?	N/A
tsplink function	
tsplink.node	N/A
tsplink.reset	N/A
tsplink.state	N/A
tsplink.trigger[N].assert	N/A
tsplink.trigger[N].clear	N/A
tsplink.trigger[N].mode	N/A
tsplink.trigger[N].overrun	N/A
tsplink.trigger[N].release	N/A
tsplink.trigger[N].stimulus	N/A
tsplink.trigger[N].wait where N is 1 to 3	N/A
userstring.add	N/A
userstring.catalog	N/A
userstring.delete	N/A
userstring.get	N/A
upgrade	
upgrade.unit	N/A
waitcomplete	
waitcomplete	*WAI

KEITHLEY

A GREATER MEASURE OF CONFIDENCE

KEITHLEY INSTRUMENTS, INC. ■ 28775 AURORA ROAD ■ CLEVELAND, OHIO 44139-1891 ■ 440-248-0400 ■ Fax: 440-248-6168 ■ 1-888-KEITHLEY ■ www.keithley.com

BELGIUM
Sint-Pieters-Leeuw
Ph: 02-363 00 40
Fax: 02-363 00 64
www.keithley.nl

CHINA
Beijing
Ph: 8610-82255010
Fax: 8610-82255018
www.keithley.com.cn

FINLAND
Espoo
Ph: 09-88171661
Fax: 09-88171662
www.keithley.com

FRANCE
Saint-Aubin
Ph: 01-64 53 20 20
Fax: 01-60-11-77-26
www.keithley.fr

GERMANY
Germering
Ph: 089-84 93 07-40
Fax: 089-84 93 07-34
www.keithley.de

INDIA
Bangalore
Ph: 080-26771071-73
Fax: 080-26771076
www.keithley.com

ITALY
Milano
Ph: 02-553842.1
Fax: 02-55384228
www.keithley.it

JAPAN
Tokyo
Ph: 81-3-5733-7555
Fax: 81-3-5733-7556
www.keithley.jp

KOREA
Seoul
Ph: 82-2-574-7778
Fax: 82-2-574-7838
www.keithley.co.kr

MALAYSIA
Kuala Lumpur
Ph: 60-3-4041-0899
Fax: 60-3-4042-0899
www.keithley.com

NETHERLANDS
Gorinchem
Ph: 0183-63 53 33
Fax: 0183-63 08 21
www.keithley.nl

SINGAPORE
Singapore
Ph: 65-6747-9077
Fax: 65-6747-2991
www.keithley.com.sg

SWEDEN
Solna
Ph: 08-50 90 46 00
Fax: 08-655 26 10
www.keithley.com

SWITZERLAND
Zürich
Ph: 044-821 94 44
Fax: 41-44-820 30 81
www.keithley.ch

TAIWAN
Hsinchu
Ph: 886-3-572-9077
Fax: 886-3-572-9031
www.keithley.com.tw

UNITED KINGDOM
Theale
Ph: 0118-929 75 00
Fax: 0118-929 75 19
www.keithley.co.uk

Printed in the U.S.A. No. 2875 1007