www.tek.com/keithley

Automated Characterization Suite (ACS)

Programmer's Manual

ACS-907-01 Rev. H November 2023

ACS-907-01H

KEITHLEY
A Tektronix Company

Automated Characterization Suite (ACS)

Programmer's Manual

These are the original instructions in English.

Document number: ACS-907-01 Rev. H November 2023

KEITHLEY
A Tektronix Company

Safety precautions

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

This product is intended for use by personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the user documentation for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product warranty may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the user documentation. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, perform safe installations, and repair products. Only properly trained service personnel may perform installation and service procedures.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

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

Safety precaution revision as of June 2018.

Table of contents

Section 1

ACS programming overview

In this section:

Test modules introduction

ACS Software allows you to create and sequence measurements using interactive, script, and Python® test modules.

Interactive test modules (ITMs) interactively define tests in the ITM by assigning instrument resources to the pads of a device under test (DUT). The ITMs are configured for what each instrument will force and measure by entering information in the fields for the ITM graphical user interface (GUI).

Script test modules (STMs) are used to control the instruments. When you create a test, a script executes on the source measure unit (SMU). The script makes calls to the Keithley Instruments Test Script Processor (TSP) Library or Linear Parametric Test Library (LPTLib) and to the Lua programming language statements. Executing the script on the instrument allows you to run your test code at the fastest possible speed.

Python test modules (PTMs) create modules using calls to the LPTLib. Unlike STMs, PTM modules use the Python programming language and do not execute on the instrument. PTM modules are supported by all the instruments that ACS supports, including Keithley Instruments and external instruments connected through unified VISA communications using GPIB, LAN, or USB. An added feature of creating PTM tests is the ability to create test modules with custom user interfaces.

ACS programming methods

C-language test modules (CTMs) are created using Microsoft Visual C++. This method of test creation is useful when interfacing with external DLL libraries or when ACS is running directly on a Keithley 4200A-SCS. There is no access to the LPTLib when this method is used. As a result, this is the most difficult method of test library creation and should be reserved for use in creating drivers for unsupported instruments or incorporating third-party libraries into the ACS environment.

This manual describes how to use the Keithley LPTLib instrument functions and introduces how to create Python® test modules (PTMs) and script test modules (STMs). It does not describe how to create CTM or interactive test module (ITM) tests. Additionally, it does not describe how to program using the Python or Lua programming languages. The following list references Python and Lua programming resources:

- lua.org
- *Programming in Lua*, 2nd Edition, Roberto Ierusalimschy, Lua.org (publisher)
- www.python.org
- www.linuxjournal.com/article/3946
- *Programming Python*, 4th Edition, Mark Lutz, O'Reilly Media (publisher)

Create PTM or STM test libraries and modules

The ACS Script Editor is used to create a Python® (PTM) test library or script (STM) test library.

NOTE

PTM tests are flexible and relatively simple to create because they do not run exclusively on the instruments. They are, however, slower than equivalent STM modules because the commands are sent one at a time to the instruments.

NOTE

To create an STM library, you follow a similar series of steps. The only differences are that you select **TSP** in the Script Editor, use the Lua programming language, and import an STM module to your test project.

To create a PTM or STM test library module:

1. In the Tools menu, select **Script Editor** to open the editor.

Figure 1: Script Editor in Tools menu

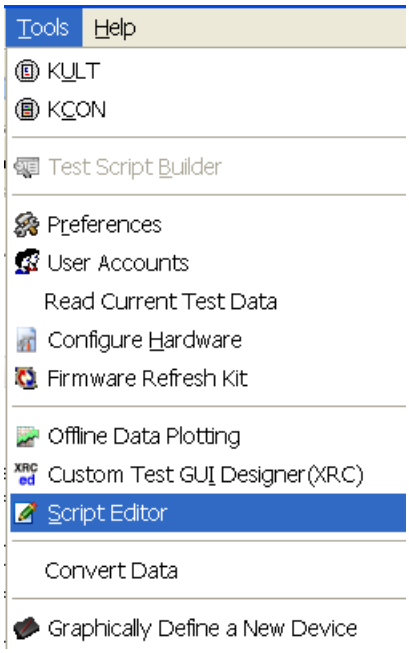
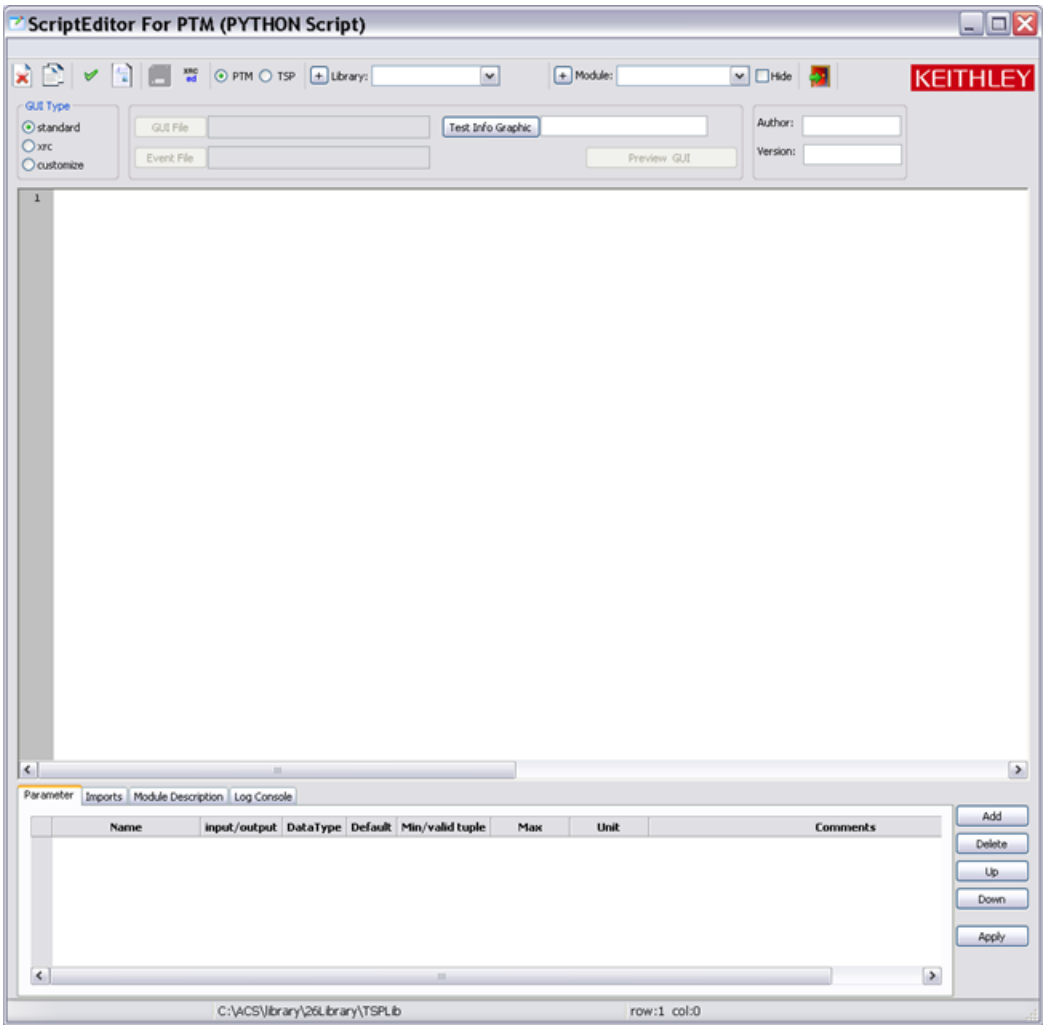


Figure 2: Script Editor for PTM



2. Select the test library type:
   - **Python test module:** Select **PTM**
   - **Script test module:** Select **TSP**.

   In this example, PTM is selected.

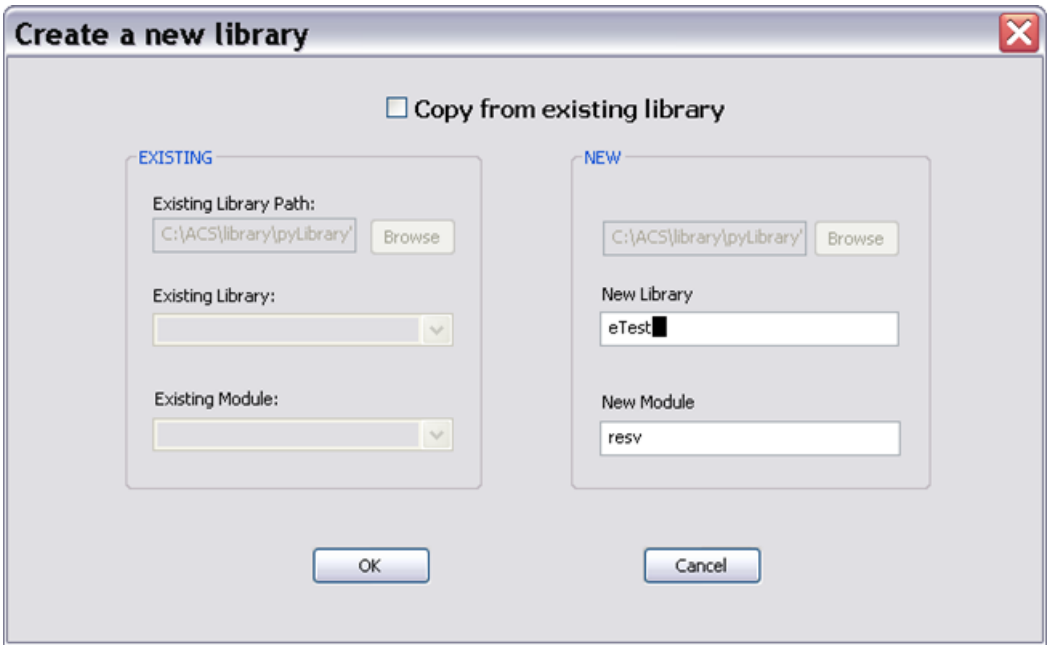
Figure 3: Test library types

3. Create a new library or add to an existing library. To create a new library, select the **+** Library icon.

Figure 4: Add Library icon on toolbar

The create a new library dialog box is displayed.

Figure 5: Create a new library dialog



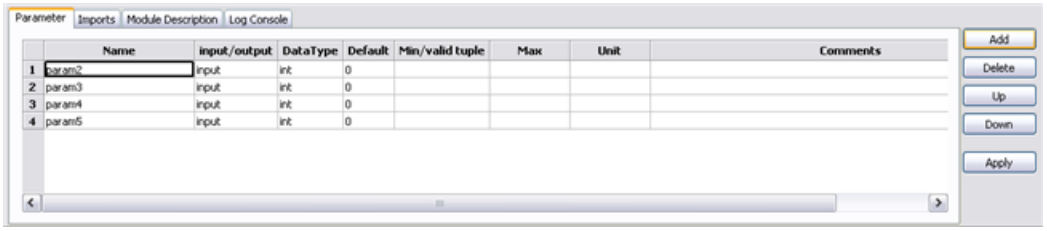
4. Under **New Library**, enter `eTest` as the name for the new library.
5. Under **New Module**, enter `resv` as the name of the first module in the new library.
6. Select **OK**. The Script Editor shows the name of the newly created library and first test module in the Library and Module fields, as shown in the following figure.

Figure 6: Library and Module names

For this example, a module to measure resistance is created (`resv`). The resistance module has three input values: High pin (`hpin`), low pin (`lpin`), and voltage force (`vforce`). The module also returns the calculated resistance.

7. To create the input and output values, select the **Parameter** tab at the bottom of the Script Editor and select **Add**. For this example, Add was selected four times, once for each input and output parameter.

Figure 7: Script Editor Parameter tab



8. Enter the names of the parameters as shown in the following figure. Select the **input/output** field next to each parameter name and select **input**. For resistance, select **output**. Pins are integer values. Select **Data Type** as shown for each input parameter. Output parameters cannot be selected.

Figure 8: Names of parameters

	Name	input/output	DataType	Default	Min/valid tuple	Max	Unit
1	hpin	input	int	0			
2	lpin	input	int	0			
3	vforce	input	int	0			
4	resistance	output	str	resistance			

9. Select **Apply** when done. A Python function declaration is automatically created in the edit area of the Script Editor.
10. Create the test code. Each Python module must include imports that enable the test module to locate and use the LPTLib and the ACS data-handling functions. To do this, select the **Imports** tab and type the text shown in the following figure.

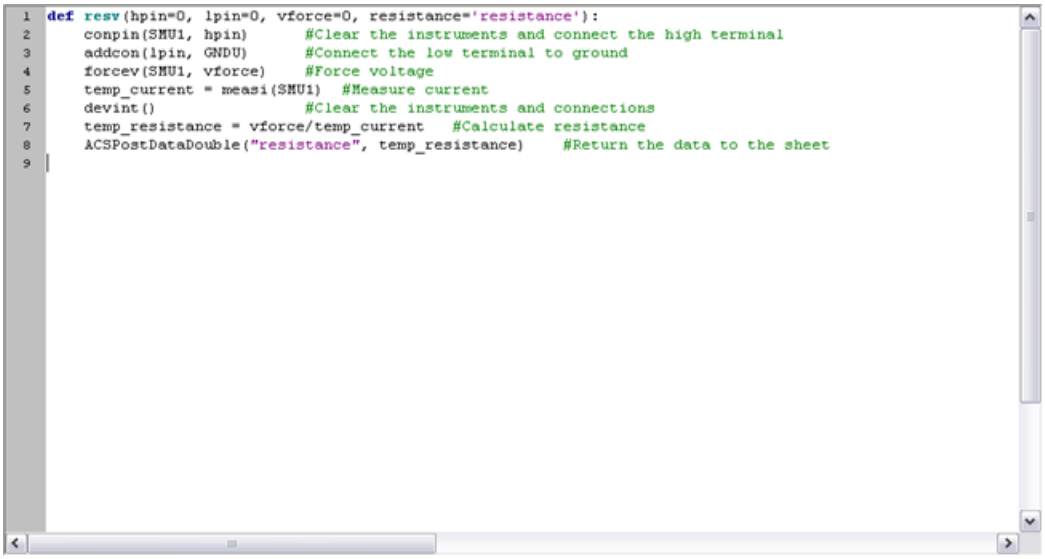
Figure 9: Script Editor Imports tab

11. Select **Apply**.
12. For the `resv` module, copy the test code below to Python. This test code does the following:
    - Uses the switch matrix to connect the pins to the SMUs
    - Forces voltage using the SMU
    - Measures current using the SMU
    - Calculates the resistance using Python
    - Returns the data to ACS

```
def resv(hpin=0, lpin=0, vforce=0, resistance="resistance"):
   #Clear the instruments and connect the high terminal.
   conpin(SMU1, hpin)           #Return the data to the sheet.
   addcon(lpin, GNDU)           #Connect the low terminal to ground.
   forcev(SMU1, vforce)         #Force voltage.
   temp_current = measi(SMU1)   #Measure current.
   devint()                     #Clear the instruments and connections.
   temp_resistance = vforce/temp_current  #Calculate resistance.
   ACSPostDataDouble("resistance", temp_resistance) #Return the data to
      the sheet.
```

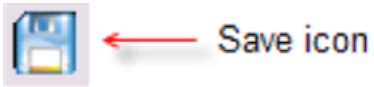
Python is a positional language, so the editor automatically indents each statement one tab stop, as shown in the following figure.

Figure 10: Script Editor resv module information



13. On the ACS Script Editor toolbar, select **Save**.

Figure 11: Script Editor Save icon

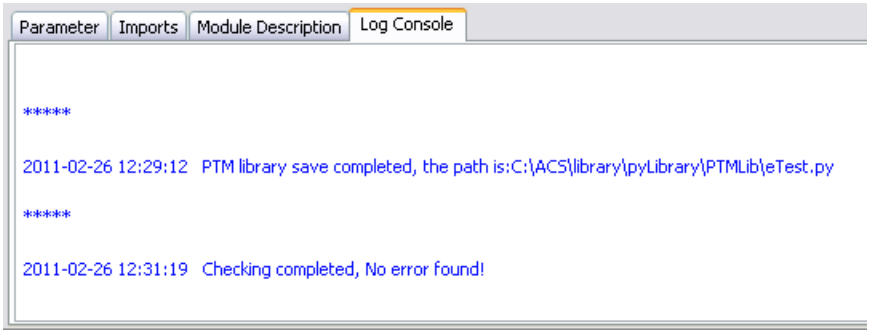


14. On the ACS Script Editor toolbar, select the **check** function, shown in the following figure, to scan for syntax errors.

Figure 12: Script Editor check function

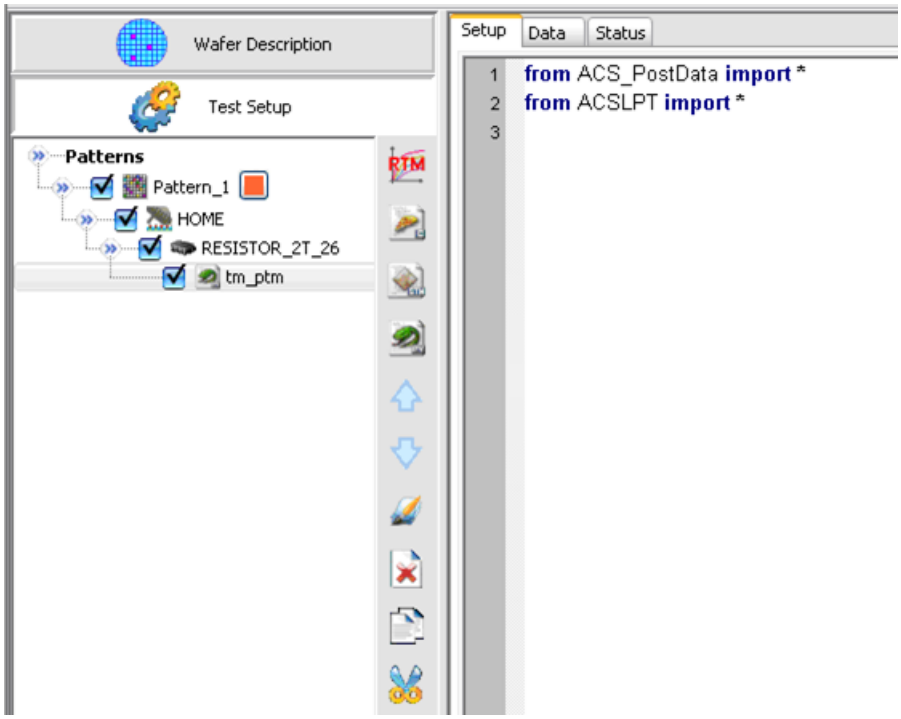
If there are no errors, a message in the Log Console tab opens, as shown in the following figure. The newly created library and module are now ready.

Figure 13: Script Editor Log Console tab



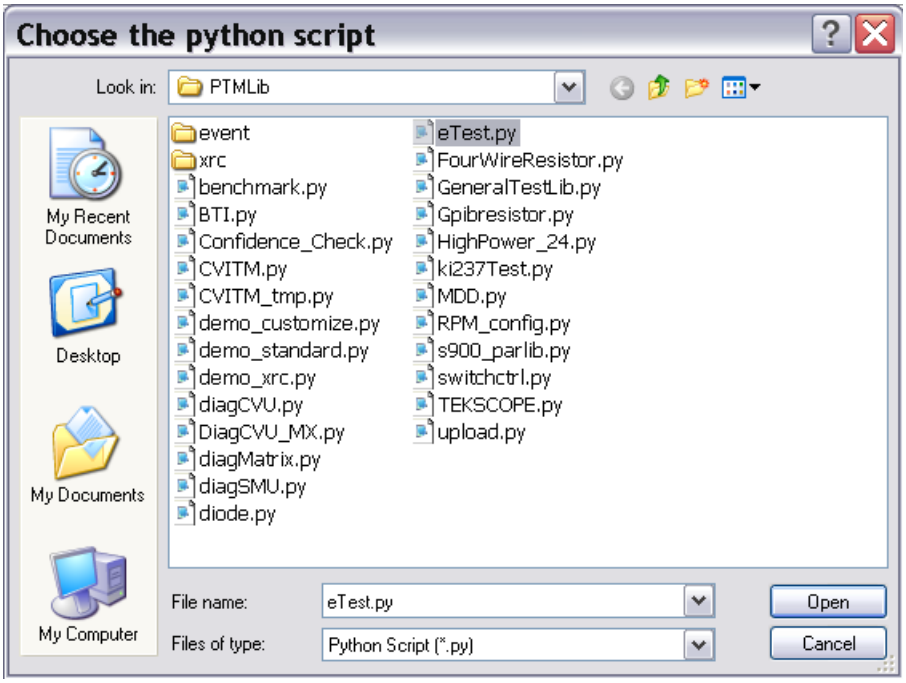
15. To use the module, add a new PTM to the ACS project.

Figure 14: ACS PTM project



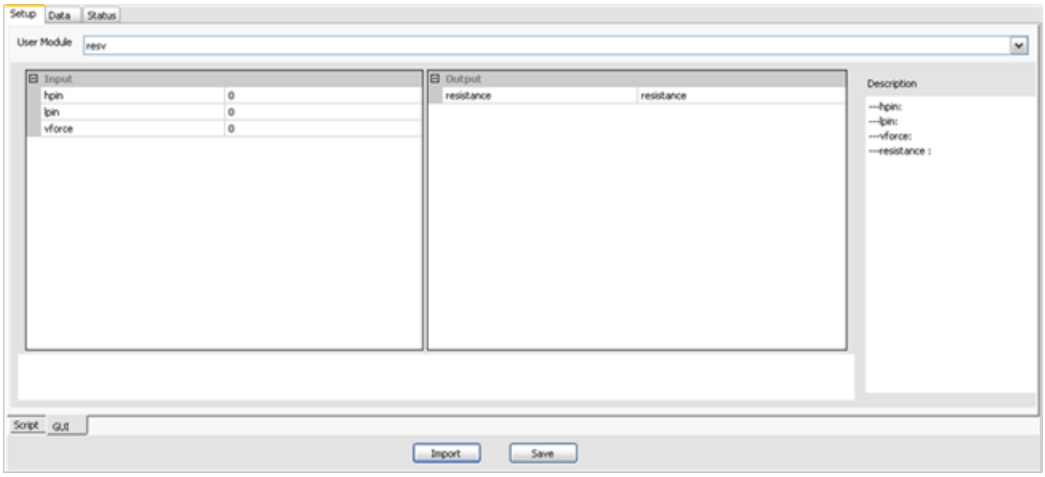
16. From the ACS Setup tab, select **Import**. The Choose python script dialog box opens.
17. Select the library to import from the list. For this example, select `eTest.py` and select **Open**.

Figure 15: ACS Choose the Python script



18. After you open the Python script, the ACS Setup tab displays an input form for the newly created test module. To use the module, input the pin numbers and force voltage to use and select **Save**.

Figure 16: ACS Setup tab GUI



The module for the test project is ready for use. If you added more than one module to the library, select the correct module by selecting the down arrow in the user module field and select a different module.

Create a custom formula

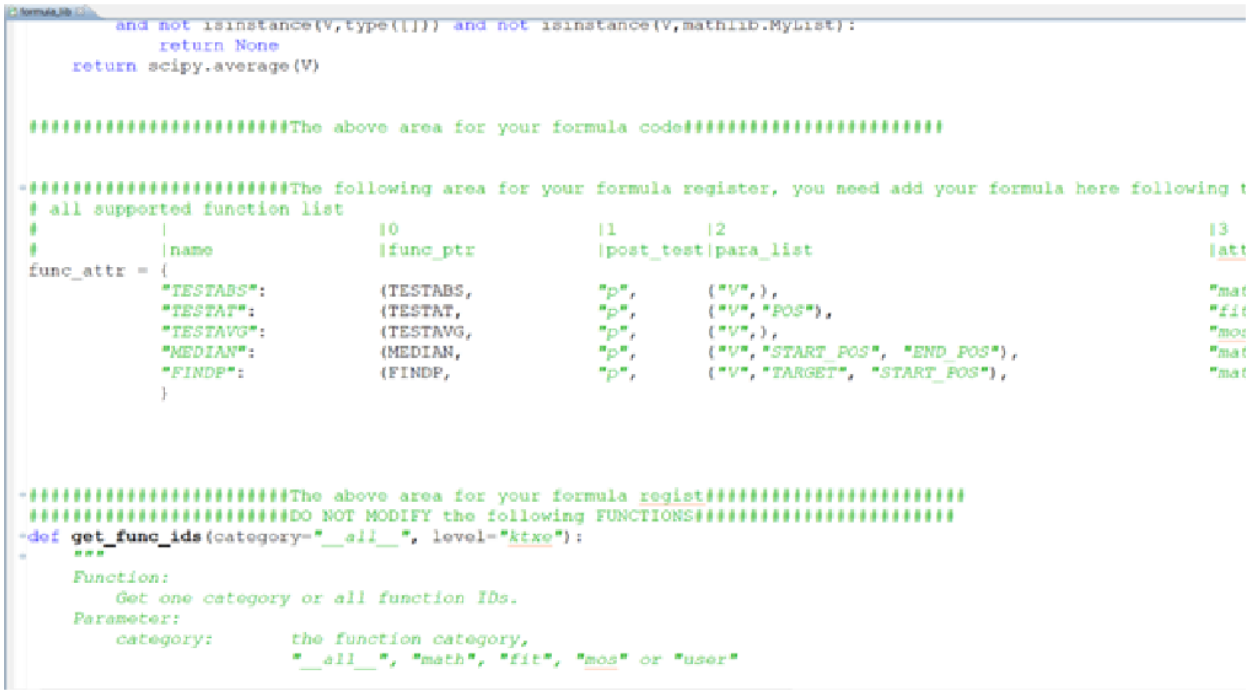
You can create your own formula and use it in the Formulator of the data panel for the test modules. Custom formulas allow you to run calculations on your test data.

To create a custom formula, access the library file in `C:\ACS\library\ExtLibrary\formula_lib.py`. You must add the formula function and the element formula in the formula register dictionary in a specific format. The `formula_lib.py` file contains the following examples that demonstrate the correct format for a formula.

Figure 17: Formula function example

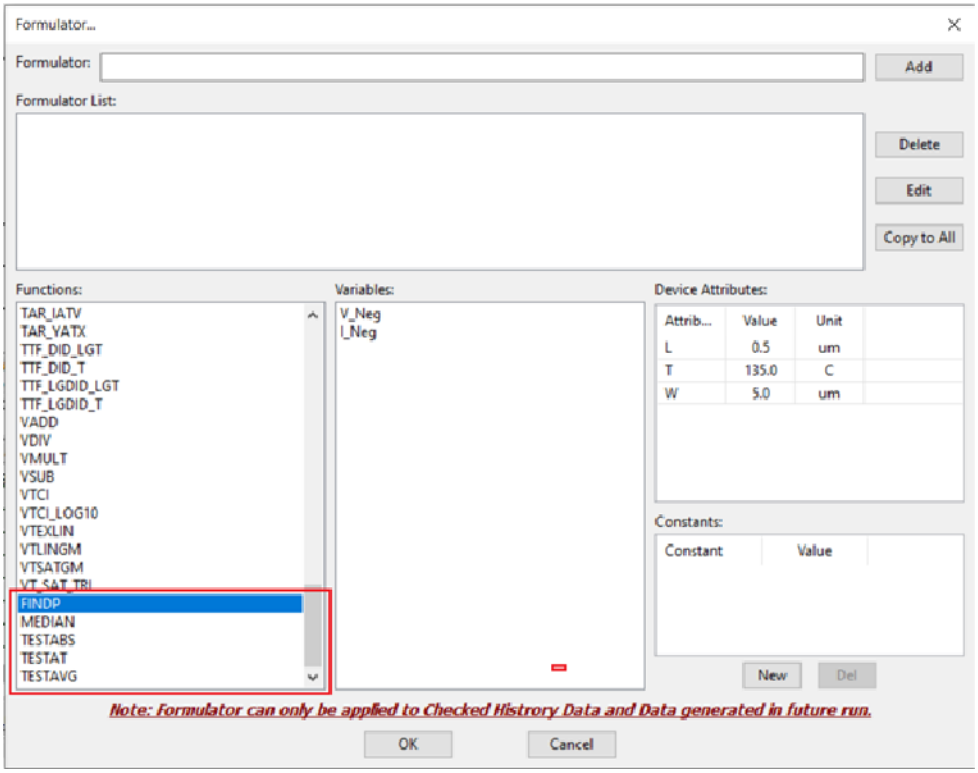
```
"""
(C)Copyright 2007 Keithley Instruments, Inc. All rights reserved.

File Name: formula_lib.py
Description: The customized formula library module.
User can write their formula here and loaded by Data Analysis formula dialog and their own model lib.

Version:    1.0
Author:     Steven
Date:       2007-10-30
Note:       func_attr -- this dictionary contains all supported functions' attribute.
            To user their formulas, user need regist them here.
"""

import types
import math
import scipy
import mathlib
from mathlib import *

##########################The following area for your formula code, you can modify and add following python syntax
def FINDP(v, x, start_pos):
    """
    Function:
        find down
    Parameter:
        v:          array to be searched
        x:          the value to be targeted
        start_pos:  position from which searching begins
    Return:
        index of the value
    Usage:
    """
```

Figure 18: Formula function dictionary example



Use the Formulator, shown in the following figure, to create your Formulator list. Set up the calculations for your test data using functions, variables, and device attributes.

Figure 19: Custom formula example



Section 2

ACS TSP LPT library reference

In this section:

TSP LPT library commands introduction

The Keithley Instruments TSP Linear Parametric Test Library (LPTLib) is a high-speed data acquisition and instrument control software library.

The TSP LPT commands in this library are programmed with Lua and can be used with script test modules (STMs). These commands are applicable to the Series 2600B and Series 2400 instruments.

NOTE

Many TSP LPT commands work with the Series 2400 System SourceMeter® instruments if the instrument is in TSP mode. Commands that are only for the Series 2600A and 2600B System SourceMeter® instruments are identified in the following command descriptions.

TSP LPT commands

The following topics list the Series 2600B TSP LPT commands in alphabetical order.

addcon

NOTE

This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command adds a terminal-pin connection.

Usage

```
addcon(*instMTRX, ter, pin, *more_pin)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Indicates additional pins to connect

Details

Terminal and pin lists must have the same number of items. Terminals and pins are matched according to the sequence. If the numbers in the terminal and pin lists are not the same, the connections are performed based on the shortest list.

Normally, the `addcon` command supports the `ROW_COLUMN` mode of the matrix. When the matrix is set to `INSTRUMENT_CARD` mode, a row is assigned automatically to connect the terminal and the pin.

For more information about how to set the `INSTRUMENT_CARD` mode and `ROW_COLUMN` mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

Example

```
addcon(MTRX1,SMU1,1)
addcon(SMU1,1)
addcon(SMU1H,1)
addcon(SMU1L,1)
addcon(SMU1,1,2,3)
addcon({SMU1,SMU2}, {1,2})
addcon(SMU1, SMU2)
addcon(CVU1H,1)
```

Also see

clrcon (on page 2-4)
conpin (on page 2-6)
delcon (on page 2-9)

addpth

NOTE

This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command adds a terminal-pin connection through a row in a matrix.

Usage

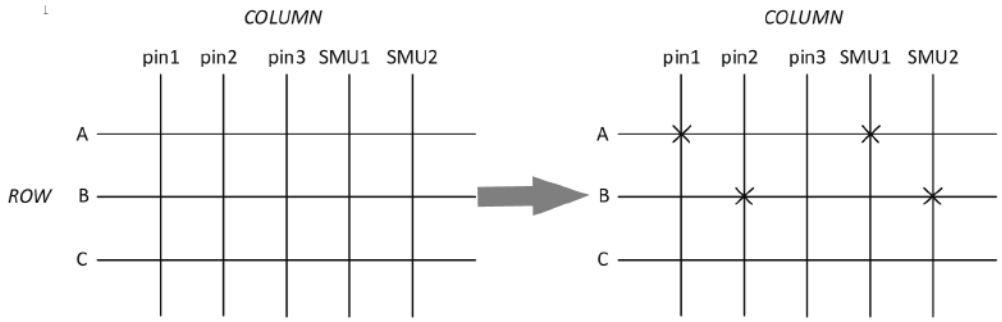
```
addpth(*instMTRX, ter, pin, row)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to be connected
pin	The list of pins to be connected
row	The row used to connect terminals and pins

Details

A terminal and a pin are connected through an assigned row to a matrix. In the following figure, terminal SMU1 and pin 1 are connected through Row A. Terminal SMU2 and pin 2 are connected through Row B. Note that you must use another instance of the `addpth` command to make a connection in another matrix.

Figure 20: addpth and conpth command example



This function can only be applied when the matrix is set to `INSTRUMENT_CARD` mode.

For more information about how to set the `INSTRUMENT_CARD` mode and `ROW_COLUMN` mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

NOTE

This TSP LPT command can only be used with a matrix connected using TSP-Link®.

Example 1

```
addpth(MTRX1,SMU1,1,'A')
addpth(SMU1,1,'A')
addpth(SMU1H,1,'A')
addpth(SMU1L,1,'A')
```

This is an example of using the `addpth` command with a Model 707/708 Switch Matrix.

Example 2

```
addpth(MTRX1,SMU1,1,'1')
addpth(SMU1,1,'1')
addpth(SMU1H,1,'1')
addpth(SMU1L,1,'1')
```

This is an example of using the `addpth` command with a Series 3700 System Switch.

Also see

clrcon (on page 2-4)
conpth (on page 2-7)
delpth (on page 2-10)

avgi/avgv

This command performs a series of measurements and averages the results.

Usage

```
avgi(SMUX, Itable, step_num, step_time)
avgv(SMUX, Vtable, step_num, step_time)
```

`SMUX`	The instrument identification code of the instrument; for example, `SMU1`
`Itable`	The table you created; the measured current value is saved into `Itable[1]`
`Vtable`	The table you created; the measured voltage value is saved into `Vtable[1]`
`step_num`	The number of steps averaged in the measurement; this number ranges from `1` to `160,000`
`step_time`	The interval in seconds between each measurement; minimum practical time is approximately 0.0001 s (NPLC must be set as 0.001)

Also see

None

clrcon

NOTE

This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command clears all the connections of all the matrices or specified matrices.

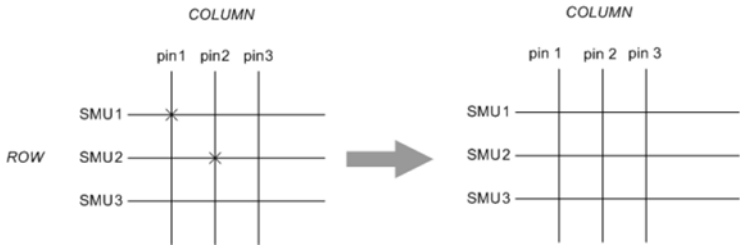
Usage

```
clrcon(unitname)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf, such as MTRX1

Details

Figure 21: clrcon (clear all) connections example



Example

```
clrcon()
clrcon(MTRX1)
```

Clear all matrix connections in the hardware configuration.
Clear only matrix1 connections.

Also see

addcon (on page 2-2)
conpin (on page 2-6)
delcon (on page 2-9)
devclr (on page 2-11)

clrscn

This command clears the measurement scan tables associated with a sweep.

Usage

```
clrscn()
```

Details

The measurement-scan tables associated with a sweep are internal tables defined by `savgi/savgv`, `sintgi/sintgv`, `smeasi`, or `smeasi/smeasv`, or `rtfary`. The `clrscn` command is called after the sweep and it clears these tables.

Example

```
clrscn()
```
Clear the measurement scan tables defined by the sweep.

Also see

savgi/savgv (on page 2-22)
sintgi/sintgv (on page 2-25)
smeasi/smeasv/smeast (on page 2-26)

conpin

NOTE

This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command clears existing connections and adds new terminal-pin connections.

Usage

```
conpin(*instMTRX, ter, pin, *more_pin)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Additional pins to connect

Details

Terminal and pin lists must have the same number of items and are arranged (by number) in the same sequence. If the numbers in the terminal and pin lists are not the same, the connections are performed on the shortest list and other entries are ignored.

Normally, the `conpin()` command supports the `ROW_COLUMN` mode of the matrix. When the matrix is set to `INSTRUMENT_CARD` mode, a row is assigned automatically to connect the terminals and the pins.

For more information about how to set the `INSTRUMENT_CARD` mode and `ROW_COLUMN` mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

NOTE

This TSP LPT command can only be used with a matrix connected using TSP-Link®.

Example

```
conpin(MTRX1,SMU1,1)
conpin(SMU1,1)
conpin(SMU1H,1)
conpin(SMU1L,1)
conpin(SMU1,1,2,3)
conpin({SMU1,SMU2}, {1,2})
conpin(CVU1H,1)
```

This example demonstrates different methods of using `conpin` to connect terminals to pins.

Also see

addcon (on page 2-2)
clrcon (on page 2-4)
delcon (on page 2-9)

conpth

NOTE

This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command clears all matrix connections and makes a new terminal-pin connection through a row.

Usage

`conpth(*instMTRX, ter, pin, row)`

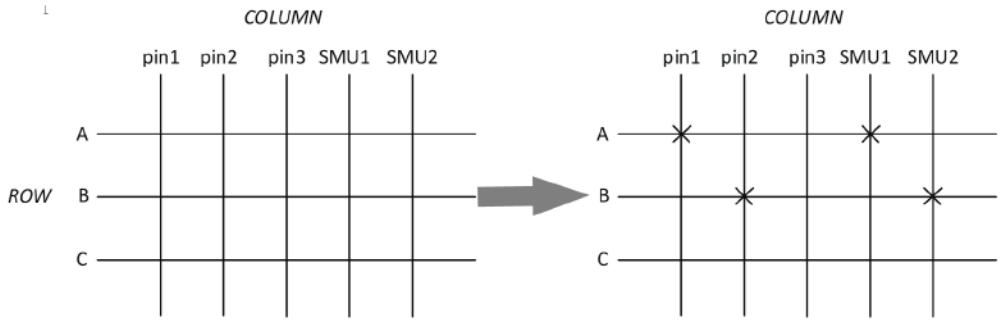
instMTRX	The matrix name in the hardware configuration (the name is optional)
ter	The list of terminals to connect
pin	The list of pins to connect
row	The row used to connect terminals and pins

Details

The terminal and pin are connected through an assigned row in a matrix. In the following figure, terminal SMU1 and pin1 are connected through row A. Terminal SMU2 and pin2 are connected through row B.

You must use one instance of the conpth command for each connection made in a matrix. You cannot use the conpth command to make a connection in multiple matrices (see the following figure).

Figure 22: addpth and conpth command example



You must use one instance of the *conpth* command for each connection made in a matrix. You cannot use the *conpth* command to make a connection in multiple matrices (see the following figure).

This function can only be applied when the matrix is set to INSTRUMENT_CARD mode.

For more information about how to set the INSTRUMENT_CARD mode and ROW_COLUMN mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

NOTE

This TSP LPT command can only be used with a matrix connected using TSP-Link®.

Example 1

```
conpth(MTRX1,SMU1,1,'A')
conpth(SMU1,1,'A')
conpth(SMU1H,1,'A')
conpth(SMU1L,1,'A')
```

This is an example of using the conpth command with a Model 707B or 708B Switch Matrix.

Example 2

```
conpth(MTRX1,SMU1,1,'1')
conpth(SMU1,1,'1')
conpth(SMU1H,1,'1')
conpth(SMU1L,1,'1')
```

This is an example of using the conpth command with a Series 3700A System Switch.

Also see

addpth (on page 2-3)
clrcon (on page 2-4)
delpth (on page 2-10)

crtbf

This command applies to only Series 2600B System SourceMeter® instruments. This command creates a buffer in which the specified SMU stores its measurements.

Usage

```
buff_name = crtbf(SMUX, buff_cap, append_flag, timestamp_flag)
```

buff_name	The name of the buffer
SMUX	The instrument identification code of the instrument; for example, `SMU1`
buff_cap	The capacity of the buffer
append_flag	Enable or disable buffer append mode: ■ To enable: Set to `KI_EBAP` ■ To disable: Set to `KI_DBAP`
timestamp_flag	Enable or disable timestamp collection: ■ To enable: Send `KI_EBTS` ■ To disable: Send `KI_DBTS`

Also see

None

delcon

> **NOTE**
> This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command deletes a terminal-pin connection.

Usage

```
delcon(*instMTRX, ter, pin, *more_pin)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Indicates additional pins to connect

Details

The terminal and pin must be connected before the command will delete the terminal pin connection, otherwise the connection is invalid.

> **NOTE**
> This TSP LPT command can only be used in a matrix connected using TSP-Link(R).

Example

```
delcon(MTRX1,SMU1,1)
delcon(SMU1,1)
delcon(SMU1H,1)
delcon(SMU1L,1)
delcon(SMU1,1,2,3)
delcon({SMU1,SMU2}, {1,2})
```

This example shows valid methods for defining the matrix, terminals, and pins.

Also see

addcon (on page 2-2)
clrcon (on page 2-4)
conpin (on page 2-6)

delpth

> **NOTE**
> This TSP LPT command can only be used on a matrix connected with TSP-Link®.

This command deletes a terminal-pin connection based on a specific row in a matrix.

Usage

```
delpth(*instMTRX, ter, pin, row)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to disconnect
pin	The list of pins to be connected
row	The row used to connect terminals and pins

Details

The terminal, pin, and row must be in the same group when connected. Otherwise, the command is not valid.

> **NOTE**
> This TSP LPT command can only be used with a matrix connected using TSP-Link®.

Example 1

```
delpth(MTRX1,SMU1,1,'A')
delpth(MTRX1,1,'A')
delpth(SMU1H,1,'A')
delpth(SMU1L,1,'A')
```

This is an example of using the `delpth` command with a Model 707B or 708B Switch Matrix.

Example 2

```
delpth(MTRX1,SMU1,1,'1')
delpth(SMU1,1,'1')
delpth(SMU1H,1,'1')
delpth(SMU1L,1,'1')
```

This is an example of using the `delpth` command with a Series 3700A System Switch.

Also see

addpth (on page 2-3)
clrcon (on page 2-4)
conpth (on page 2-7)

devclr

This command sets all sources to a zero state.

Usage

```
devclr()
```

Example

```
devclr()
```

Set all sources to a zero state.

Also see

None

devint

This command resets all instruments and clears the system by opening all relays and disconnecting the pathways.

Usage

```
devint()
```

Details

This command resets all the instruments in the TSP-Link group to their default states. The default states for each instrument are provided in the instrument documentation.

The following tables provide some of the default settings for the Series 2400 and the 2600B instruments after applying the `devint` LPT command.

Default settings	
Series 2400 SourceMeter® instrument settings after `devint` is applied	
Output	Turns source off
Reset	Resets all bits of the following registers to 0: ■ Standard event register ■ Operation event register ■ Measurement event register ■ Questionable event register
Long-form and short-form versions	Command word is sent in short-form version
Terminals	Setting before `devint` was called

Default settings	
Series 2600B SourceMeter® instrument settings after `devint` is applied	
Current range	0.1 A (all SMUs)
Output	Clear
Error queue	Clear
Status model	Reset all bits
Error display	Disable
PLC	0.001
	1 (for intgx)
Measure count	1
DTNS	Clear sweep table, clear trigger, clear garbage, then set all 26xx in DTNS group 0
Sense mode	Depends on ACS software preference setting
Reset	The default factory setting for each instrument

This command also performs the following actions before resetting the instruments:

- Clears all sources by calling `devclr`
- Clears the matrix cross-points by calling `clrcon`
- Clears the sweep tables by calling `clrscn`

Example

```
devint()
```

Resets all instruments and clears the system.

Also see

clrcon (on page 2-4)
clrscn (on page 2-5)
devclr (on page 2-11)

disable

This command stops the timer, sets the time value to zero, and stops the timer reading.

Usage

```
disable(ntimer[Y])
```

Y	Timer number, such as 1, 2, 3

Example

```
disable(ntimer[1])
```

This example disables a timer named ntimer[1].

Also see

enable (on page 2-13)

enable

This command enables a timer, which can be used with the command `meast()` to measure time.

Usage

```
enable(ntimer[Y])
```

Y	Timer number, such as 1, 2, 3

Details

The timer enabled can be used in `meast()` to measure time and can be disabled using `disable()`.

Example

```
--Enable timer ntimer[1].
enable(ntimer[1])
```

This example enables a timer named `ntimer[1]`.

Also see

disable (on page 2-13)
measi/measv/meast (on page 2-17)

forceclr

This command turns the source output off on the specified SMU.

Usage

```
forceclr(SMUX)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`

Example

```
--Turns the SMU1 source output off.
forceclr(SMU1)
```

Turns the source output of SMU1 off.

Also see

None

forcei/forcev

This command programs a sourcing instrument to generate a voltage or current at a specific level.

Usage

```
forcei(SMUX, value)
forcev(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The level of current or voltage forced in amperes or volts

Example

```
--force current 1e-6A on SMU1
forcei(SMU1, 1e-6)
--force voltage 0.02V on SMU2
forcev(SMU2, 0.02)
```

Sets up SMU1 to source 1 µA and SMU2 to source 0.02 V.

Also see

None

intgi/intgv

This command makes voltage or current measurements averaged over a user-defined period (usually one AC-line cycle).

Usage

```
intgi(SMUX, Itable)
intgv(SMUX, Vtable)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
Itable	The table created by you; the measured current value is saved into `Itable[1]`
Vtable	The table created by you; the measured voltage value is saved into `Vtable[1]`

Details

Averaging is done in the hardware by integration of the analog measurement signal over a specified time. The integration is automatically corrected for 50 Hz or 60 Hz power mains.

Example

```
--force current 1e-6A on SMU1
forcei(SMU1, 1e-6)
```

Also see

None

ioli/iolv/ioliv

This command applies to only Series 2600B System SourceMeter® instruments.

This command uses overlap mode to measure current, voltage, or current and voltage.

Usage

```
ioli(SMUX, i_buff_name)
iolv(SMUX, v_buff_name)
ioliv(SMUX, i_buff_name, v_buff_name)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
i_buff_name	The buffer to store current measurements; see **Details**
v_buff_name	The buffer to store current measurements; see **Details**

Details

The integration time is set by `setmode()` and the measure count is set by `setcount()`. The only difference between this function and `msoli()` is that the integration time `msoli()` uses a fixed 0.001 NPLC.

The buffer must be created by `crtbf()` for the same SMU.

Also see

crtbf (on page 2-9)

limiti/limitv

This command allows you to specify a current or voltage limit.

Usage

```
limiti(SMUX, value)
limitv(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The value of current limit in amperes or the voltage limit voltage

Example

```
--Set the current limit of SMU1 to 0.01 A.
limiti(SMU1, 0.01)
--Set voltage limit of SMU2 to 200 V>.
limitv(SMU2, 200)
```

Set the current limit for SMU1 to 0.01 A and the voltage limit of SMU2 to 200 V.

Also see

None

lorangei/lorangev

This command defines the lowest autorange limit for current or voltage measurements.

Usage

```
lorangei(SMUX, value)
lorangev(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The value of the lowest autorange limit in amperes or volts

Example

```
--Set lowest autorange limit of current on SMU1 to 1E-6A.
lorangei(SMU1, 1E-6)
--Set lowest autorange limit of voltage on SMU2 to 0.2V.
lorangev(SMU2, 0.2)
```

Sets the lowest autorange limit for current on SMU1 to 1 µA.
Sets the lowest autorange limit for voltage on SMU2 to 0.2 V.

Also see

None

measi/measv/meast

This command allows the measurement of voltage, current, or time.

Usage

```
measi(SMUX, Itable_num)
measv(SMUX, Vtable)
meast(ntimer[Y], Ttable)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
Itable_num	The table created by you; the measured current value is saved into `Itable[1]`
Vtable	The table created by you; the measured voltage value is saved into `Vtable[1]`
ntimer[Y]	The timer number
Ttable	The table created by you; the measured voltage value is saved into `Ttable[1]`

Also see

None

msoli/msolv/msoliv

> **NOTE**
>
> Only for Series 2600B System SourceMeter® instruments.

This command measures current, voltage, or current and voltage using overlap mode with a fixed 0.001 NPLC.

Usage

```
msoli(SMUX, i_buff_name)
msolv(SMUX, v_buff_name)
msoliv(i_buff_name, v_buff_name)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
i_buff_name	The buffer to store current measurements; see **Details**
v_buff_name	The buffer to store voltage measurements; see **Details**

Details

The buffer must be created by `crtbf()` for the same SMU.

Also see

None

postscript

This command prints a list of scripts that are presently stored in the parent of the Series 2600B instruments.

Usage

```
postscript(location)
```

location	The location of the script: ■ Volatile memory: `0` ■ Nonvolatile memory (default): `1`

Also see

None

postbuffer

This command prints buffered data to a GPIB output buffer in binary format.

Usage

```
postbuffer("name", start_index, end_index, buff_name, avg_num)
```

name	A string that represents the values in the script, defined by the script writer
start_index	The starting index of values to post and print
end_index	The ending index of values to post and print
buff_name	The name of the buffer to print; it could be a default name or a user-defined name
avg_num	The average number (must be an integer); if this number is equal to 2 or greater, the DATA Engine automatically calculates the average result of each `avg_num` value; if this parameter is not provided, the system assigns a default value of `1` (print every value point)

Details

ACS software can only recognize buffered data printed by the `postbuffer` function.

Also see

None

postbuftime

This command prints timestamps of buffered data in binary format.

Usage

```
postbuftime("name", start_index, end_index, buff_name, avg_num)
```

name	A string that represents the values in the script
start_index	The starting index of values to post and print
end_index	The ending index of values to post and print
buff_name	The name of the buffer to print; it could be a default name or a user-defined name
avg_num	The average number (must be an integer); if this number is equal to 2 or greater, the DATA Engine automatically calculates the average result of each `avg_num` value; if this parameter is not provided, the system assigns a default value of `1`

Details

ACS software only recognizes buffered `timestamp` data printed by the `postbuftime` function.

In the same buffer, always use `avg_num` with `postbuffer()`, or the number of the timestamps will not match the number of the values. If you do not define this parameter, the system uses a default value of 1 (print every value point).

Also see

None

postdata

This command prints a single value.

Usage

```
postdata("name", value)
```

name	A string that represents the values in the script
value	The value to print

Details

The value to print, for example, could be an execution: "`node[2].smua.measure.i()`", or "`measi(SMU1)`".

ACS software only recognizes single values printed by the `postdata` function.

Also see

None

posterror

This command prints all errors in the error queue separately.

Usage

```
posterror()
```

Details

This function was designed for the DATA Class engine of ACS.

Also see

None

postglobal

This command prints all global variables in real-time memory of the Series 2600B.

Usage

```
postglobal()
```

Also see

None

postsmuinfo

This command prints information for all SMUs.

Usage

```
postsmuinfo()
```

Also see

None

posttable

This command prints the data in a specified table. Each item in the table must be a numeric value.

Usage

```
posttable("name", tableName)
```

name	A string that represents the values in the script, defined by the script writer
tableName	The name of the table

Also see

None

rangei/rangev

This command selects the current or voltage measurement range and prevents the selected instrument from autoranging.

Usage

```
rangei(SMUX, value)
rangev(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The value of the current or voltage measurement range in amperes or volts.

Details

By selecting a range, the time required for autoranging is eliminated.

Example

```
--Set the current measurement range on SMU1 to 1e-3A.
rangei(SMU1, 1e-3)
--Set the voltage measurement range on SMU2 to 20V.
rangev(SMU2, 20)
```

Set the current measurement range on SMU1 to 1 mA and the voltage measurement range on SMU2 to 20 V.

Also see

None

rdelay

This command provides a user-programmable delay in a test sequence.

Usage

```
rdelay(dDelayTime)
```

dDelayTime	The delay duration in seconds

Details

This command delays the test sequence by a specified number of seconds.

Example

```
--delay 0.001 s
rdelay(0.001)
```

Add a 1 ms delay.

Also see

None

savgi/savgv

NOTE

Only for Series 2600B System SourceMeter® instruments.

This command performs a current or voltage measurement for every point in a sweep and determines the average for every point.

Usage

```
savgi(SMUX, Itable, step_num, step_time)
savgv(SMUX, Vtable, step_num, step_time)
```

X	SMU number, such as `1`, `2`, or `3`
Itable	The table created by you; the measured current value is saved into `Itable[1]`.
Vtable	The table created by you; the measured voltage value is saved into `Vtable[1]`.
step_num	The number of measurements made at each point before the average is computed.
step_time	The time delay in seconds between each measurement within a given ramp step.

Also see

None

scnmeas

NOTE

Only for Series 2600B System SourceMeter® instruments.

This command performs a single measurement on multiple instruments at the same time.

Usage

```
scnmeas()
```

Details

This function behaves like a single point sweep. It performs a single measurement on multiple instruments at the same time. Any forcing or delaying must be done before calling `scnmeas`. You must use `smeasx`, `sintgx`, or `savgx` to set up result array, the same as setting up a sweep call. Each call to `scnmeas` adds one element to the end of each array. Calls to `scnmeas` may be mixed with calls to `sweepx`. All results are appended to the result arrays in the same way multiple `sweepx` calls behave.

Also see

None

setauto

This command sets the SMU measurement autorange.

Usage

```
setauto(SMUX)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`

Also see

None

setcount

This command applies to only Series 2600B System SourceMeter® instruments. This command sets the number of measurements performed when a measurement is requested.

Usage

```
setcount(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	Number of measurements

Details

This attribute controls the number of measurements taken any time a measurement is requested. When using a reading buffer with a measure command, the count also controls the number of readings to be stored.

The reset function sets the measure count to 1.

Example

```
--Set the number of measurements on SMU1 to 5.
setcount(SMU1, 5)
```

Set the number of measurements on SMU1 to 5.

Also see

None

setitv

This command applies to only Series 2600B System SourceMeter® instruments. This command sets the interval between multiple measurements in seconds.

Usage

```
setitv(SMUX, value)X = SMU number(1,2,3,...)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The interval between multiple measurements in seconds

Details

This attribute sets the time interval between groups of measurements when `setcount()` is set to a value greater than 1. The SMU will attempt to start the measurement of each group when scheduled.

- If the SMU cannot keep up with the interval setting, measurements are made as fast as possible.
- The reset function sets the measure interval to 0.

Example

```
--Set the interval between multiple measurements on SMU1 to 0.001.
ssetitv(SMU1, 0.001)0.001 s
```

Set the interval between multiple measurements on SMU1 to 0.001 s.

Also see

None

setmode

This command sets instrument-specific operating mode parameters.

Usage

```
setmode(SMUX)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`

Details

This will modify the parameters specific operating characteristics of the specified instrument (see the following table).

Parameters			Comments
smu[X]	**Modifier**	**Value**	
smu[X]	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time the SMU will use for the `intgx` command. The default `devint` value is 1.0. The valid range is 0.001 to 25.0.
	KI_AVGMODE	KI_MEASX	Controls what kind of readings are taken for `avgX` calls. The `devint` default value is KI_MEASX. When KI_INTEGRATE is specified, the time used is that specified by the setmode (KI_INTGPLC) call.
		KI_INTEGRATE	
	KI_OFFMODE	KI_OFF_NORM	Set source output-off mode. KI_OFF_NORM: Outputs 0V when the output is turned off. KI_OFF_ZERO: Zero the output (in either volts or current) when off. KI_OFF_OPEN: Opens the output relay when the output is turned off.
		KI_OFF_ZERO	
		KI_OFF_OPEN	
	KI_SENSE	KI_SENSE_LOCA	Sets the sense mode to remote, local, or calibration. KI_SENSE_LOCA: selects local sense (2-wire). KI_SENSE_REMO: selects remote sense (4-wire). KI_SENSE_CALA: selects calibration sense mode.
		KI_SENSE_REMO	
		KI_SENSE_CALA	

Also see

None

sintgi/sintgv

This command performs an integrated current or voltage measurement for every point in a sweep.

Usage

```
sintgi(SMUX, Itable)
sintgv(SMUX, Vtable)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
Itable	The table created by you; the measured current value is saved into `Itable[1]`
Vtable	The table created by you; the measured voltage value is saved into `Vtable[1]`

Also see

None

slorangei/slorangev

This command applies to only Series 2600B System SourceMeter® instruments. This command defines the lowest autorange limit for the current or voltage source.

Usage

```
slorangei(SMUX, value)
slorangev(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The lowest autorange limit for the current or voltage source

Example

```
--Set the lowest autorange limit for the current source on SMU1 as 1e-6 A.
slorangei(SMU1, 1e-6)
--Set the lowest autorange limit for the voltage source on SMU2 as 0.2 V.
slorangev(SMU2, 0.2)
```

Set the lowest autorange limit for the SMU1 current source to 1 µA..
Set the lowest autorange limit for the voltage source on SMU2 to 0.2 V.

Also see

None

smeasi/smeasv/smeast

This command allows several current or voltage/time measurements to be made by a specified instrument during a `sweepX` function.

Usage

```
smeasi(SMUX, Itable_num)
smeasv(SMUX, Vtable)
smeast(ntimer[Y], Ttable)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
Itable	The table created by you; the measured current value is saved into `Itable[1]`
Vtable	The table created by you; the measured voltage value is saved into `Vtable[1]`
Ttable	The table created by you; the measured voltage value is saved into `Ttable[1]`

Details

The results of the measurements are stored in the defined array.

Also see

None

srangei/srangev

This command selects the current or voltage source range and prevents the selected instrument from autoranging.

Usage

```
srangei(SMUX, value)
srangev(SMUX, value)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The current or voltage source range

Details

By selecting a range, the time required for autoranging is eliminated.

Example

```
--Set the current source range on SMU1 to 1e-3A.
srangei(SMU1, 1e-3)
--Set the voltage source range on SMU2 to 20V.
srangev(SMU2, 20)
```

Also see

None

ssetauto

This command sets the source-measure unit (SMU) source to autorange.

Usage

```
ssetauto(SMUX)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`

Also see

None

sweepi/sweepv

This command generates a ramp consisting of ascending or descending currents or voltages.

Usage

```
sweepi(SMUX, start, end, step_number, delay_time)
sweepv(SMUX, start, end, step_number, delay_time)
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`
start	The initial voltage or current level output from the sourcing instrument that is applied for the first sweep measurement; this value can be positive or negative
end	The final voltage or current level applied in the last step of the sweep; this value can be positive or negative
step_number	The number of current or voltage changes in the sweep; the actual number of forced data points is one greater than the number of steps specified
delay_time	The delay in seconds between each step and the measurements defined by the active measure list

Details

The sweep consists of a sequence of steps, each with a user-specified duration.

Also see

None

sysinit

This command sets NPLC to 0.001 and measure count to 1.

Usage

```
sysinit()
```

SMUX	The instrument identification code of the instrument; for example, `SMU1`

Details

This command affects every SMU in the system and clears error queues and resets all registers.

Also see

None

sysquery

This command queries every node and every source-measure unit (SMU) in the system and gives every SMU a unique name.

Usage

```
sysquery()
```

Details

When using this command query, SMU*X* indicates the node number and the SMU number on every Series 2600B instruments' screen.

This command sets the integration NPLC to 1 and the average mode to `KI_MEASX` on every SMU in the system.

Also see

None

LPT library command example 1

The following LPT example is provided for your reference.

Function: `R_single (sensemode, testmode, RSMU1, RSMU2, forcevalue, myLIMIT, myNPLC, testdelay, Rvalue)`

```
--[[
USRLIB INFORMATION
        VISIBILITY : NOT_HIDDEN
        GUI : True
        GroupNO : 1
        AUTHOR : John Smith
        VERSION : 1.0
        MODULES : ['R_single']
END USRLIB INFORMATION

USRLIB HELP
END USRLIB HELP
]]--

function R_single(sensemode, testmode, RSMU1, RSMU2, forcevalue, myLIMIT, myNPLC,
   testdelay, error, Rvalue)
--[[
    VISIBILITY: NOT_HIDDEN
    AUTHOR : John
    VERSION : 1.0
    TYPE : standard
    INPUTLIST
        #Name, Type, Default Value, Min Value, Max Value, Unit
        sensemode,enum,'0',('0','1'),,
        testmode,enum,'0',('0','1'),,
        RSMU1,enum,'SMU1',('SMU1','SMU2','NONE'),,
        RSMU2,enum,'GNDU',('SMU1','SMU2','GNDU','NONE'),,
        forcevalue,double,0.1,-200,200,
        myLIMIT,double,0.1,0,1,
        myNPLC,double,0.01,0,25,
        testdelay,double,0,,,
    END INPUTLIST
    OUTPUTLIST
        #Name, Type, DefaultValue, Unit
        error,str,'error',
        Rvalue,str,'Rvalue',
    END OUTPUTLIST
    DESCRIPTION
   This module is testing the value of a single resistor.
    ---sensemode: 0:KI_SENSE_LOCA, 1:KI_SENSE_REMO
    ---testmode: 0:Force v Meas i, 1:Force i Meas v
    ---RSMU1: SMU to terminal1 of resistor
    ---RSMU2: SMU to terminal2 of resistor
    ---forcevalue: SMU1 force value
    ---myLIMIT: compliance
    ---myNPLC: NPLC, 0.001~25
    ---testdelay: measure delay after stress is off
```

```
    ---error: Error status of the test
    ---Rvalue: Tested resistor value
    END DESCRIPTION
]]--
   local v_value = {}
   local i_value = {}
   local R_value = {}
   local error_value = {}

   if sensemode ~= 0 and sensemode ~= 1 then
          table.insert(error_value, -10100)
          posttable("error", error_value)
          return
   end

   if testmode ~= 0 and testmode ~= 1 then
          table.insert(error_value, -10100)
          posttable("error", error_value)
          return
   end

   --set RSMU1 NPLC
   setmode(RSMU1, KI_INTGPLC, myNPLC)
   --set RSMU1 in sensemode
   setmode(RSMU1, KI_SENSE, sensemode)

   if RSMU2 ~= GNDU then
          setmode(RSMU2, KI_SENSE, sensemode)
          --set RSMU2 current limit
          limiti(RSMU2, 1)
   end

   --if
   if testmode == 0 then
          --set RSMU1 current limit
          limiti(RSMU1, myLIMIT)
          --force RSMU1 voltage source value
          forcev(RSMU1, forcevalue)
   elseif testmode == 1 then
          --set RSMU1 voltage limit
          limitv(RSMU1, myLIMIT)
          --force RSMU1 current source value
          forcei(RSMU1, forcevalue)
   end

   --if
   if RSMU2 ~= GNDU then
          --force RSMU2 voltage source value
          forcev(RSMU2, 0)
   end

   --set delay time before measure
   delay(testdelay)
   --measure RSMU1 voltage
   intgv(RSMU1, v_value)
   --measure RSMU1 current
```

```
    intgi(RSMU1, i_value)

    R_value[1] = v_value[1]/i_value[1]
    posttable("Rvalue", R_value)

    table.insert(error_value, 0)
    posttable("error", error_value)

    devint()
end

--CALL--

-- The following code is automatically generated by the STM
    -- standard GUI ***DO NOT EDIT***

local sensemode = 0
local testmode = 0
local RSMU1 = SMU1
local RSMU2 = GNDU
local forcevalue = 0.1
local myLIMIT = 0.1
local myNPLC = 0.01
local testdelay = 0
local error = "error"
local Rvalue = "Rvalue"

R_single(sensemode,testmode,RSMU1,RSMU2,forcevalue,myLIMIT,
    myNPLC,testdelay,error, Rvalue)
```

LPT library command example 2

The following LPT example is provided for your reference.

Function: `Four_term_MOSFET_idvg (DSMU, GSMU, SSMU, BSMU, Vg_start, Vg_stop, Vg_points, Dcompliancei, Gcompliancei, Scompliancei, Bcompliancei, VD, VSS, VBULK, myNPLC, holdtime, sweepdelay, error, time, Id, Vg)`

```
--[[
USRLIB INFORMATION
        VISIBILITY : NOT_HIDDEN
        GUI : True
        GroupNO : 1
        AUTHOR :
        VERSION :
        MODULES : ['Four_term_MOSFET_idvg']
END USRLIB INFORMATION

USRLIB HELP
END USRLIB HELP
]]--

function Four_term_MOSFET_idvg(DSMU, GSMU, SSMU, BSMU, Vg_start, Vg_stop,
   Vg_points, Dcompliancei, Gcompliancei, Scompliancei, Bcompliancei, VD, VBUILK,
   VSS, myNPLC, holdtime, sweepdelay, error, time, Id, Vg)
--[[
    VISIBILITY: NOT_HIDDEN
    AUTHOR : John
    VERSION : 1.0
    TYPE : standard
    INPUTLIST
        #Name, Type, Default Value, Min Value, Max Value, Unit
        DSMU,enum,'SMU1',('SMU1','SMU2','SMU3','SMU4'),,
        GSMU,enum,'SMU2',('SMU1','SMU2','SMU3','SMU4'),,
        SSMU,enum,'GNDU',('SMU1','SMU2','SMU3','SMU4','GNDU'),,
        BSMU,enum,'GNDU',('SMU1','SMU2','SMU3','SMU4','GNDU'),,
        Vg_start,double,0,-200,200,
        Vg_stop,double,2,-200,200,
        Vg_points,double,11,1,,
        Dcompliancei,double,0.1,-1,1,
        Gcompliancei,double,0.1,-1,1,
        Scompliancei,double,0.1,-1,1,
        Bcompliancei,double,0.1,-1,1,
        VD,double,1,-200,200,
        VBUILK,double,0,-200,200,
        VSS,double,0,-200,200,
        myNPLC,double,1,0,25,
        holdtime,double,0.01,0,,
        sweepdelay,double,0.001,0,,
    END INPUTLIST
    OUTPUTLIST
        #Name, Type, DefaultValue, Unit
        error,str,'error',
```

```
        time,str,'time',
        Id,str,'Id',
        Vg,str,'Vg',
    END OUTPUTLIST
    DESCRIPTION
    ---DSMU: SMU to Drain
    ---GSMU: SMU to Gate
    ---SSMU: SMU to Source
    ---BSMU: SMU to Bulk
    ---Vg_start: Start voltage of sweep on Gate.
    ---Vg_stop: Stop voltage of sweep on Gate.
    ---Vg_points: Points of sweep on Gate.
    ---Dcompliancei: Current compliance on Drain.
    ---Gcompliancei: Current compliance on Gate.
    ---Scompliancei: Current compliance on Source.
    ---Bcompliancei: Current compliance on Bulk.
    ---VD: Voltage on Drain.
    ---VBUILK: Voltage on Bulk.
    ---VSS: Voltage on Source.
    ---myNPLC: NPLC, 0.001~25
    ---holdtime: Hold time before measurement.
    ---sweepdelay: Delay between sweep points.
    ---error: Error status of the test.
    ---time: Test time.
    ---Id: Drain current.
    ---Vg: Gate voltage.
    END DESCRIPTION
]]--
   local vg_value
   local Vg_table={}
   local Id_table={}
   local Id_value={}
   local dummy={}
   local error_value={}
   local time_value={}
   local Vg_inc
   Vg_inc = (Vg_stop - Vg_start)/(Vg_points-1)

   --set the NPLC of DSMU
          setmode(DSMU, KI_INTGPLC, myNPLC)
   --set current compliance to GSMU
   limiti(GSMU, Gcompliancei)
   --set current compliance to DSMU
   limiti(DSMU, Dcompliancei)
   --set DSMU measure range to auto
   setauto(DSMU)

          --set current compliance to SSMU
          if SSMU ~= GNDU  then
          limiti(SSMU, Scompliancei)
          --apply SSMU voltage source
          forcev(SSMU, VSS)
   end

   --if
   if BSMU ~= GNDU then
          --set current compliance to BSMU
```

```
          limiti(BSMU, Bcompliancei)
          --apply BSMU voltage source
          forcev(BSMU, VBULK)
   end

   --apply DSMU voltage source
   forcev(DSMU, VD)
   --apply GSMU voltage source
   forcev(GSMU, Vg_start)
   --set time delay before measure
   delay(holdtime)
   --perform current measure on DSMU
   intgi(DSMU, dummy)
   --apply DSMU voltage source
   forcev(DSMU, VD)
   timer.reset()

   for i=1, Vg_points do
          vg_value = Vg_start + (i-1) * Vg_inc
          --apply GSMU voltage source
          forcev(GSMU, vg_value)
          table.insert(Vg_table, vg_value)
          --set time interval between every point
          delay(sweepdelay)
          --perform current measure on DSMU
          intgi(DSMU, Id_value)
          table.insert(Id_table, Id_value[1])
          table.insert(time_value, timer.measure.t())
   end for

   table.insert(error_value, 0)
   posttable("error", error_value)

   posttable("time", time_value)
   posttable("Vg", Vg_table)
   posttable("Id", Id_table)
   devint()

end

--CALL--
--The following cod is automatically generated by the STM --standard GUI ***DO NOT
   EDIT***

local DSMU = SMU1
local GSMU = SMU2
local SSMU = GNDU
local BSMU = GNDU
local Vg_start = 0
local Vg_stop = 2
local Vg_points = 11
local Dcompliancei = 0.1
local Gcompliancei = 0.1
local Scompliancei = 0.1
local Bcompliancei = 0.1
local VD = 1
local VBUILK = 0
```

```
local VSS = 0
local myNPLC = 1
local holdtime = 0.01
local sweepdelay = 0.001
local error = "error"
local time = "time"
local Id = "Id"
local Vg = "Vg"

Four_term_MOSFET_idvg(DSMU,GSMU,SSMU,BSMU,Vg_start,
   Vg_stop,Vg_points,Dcompliancei, Gcompliancei,
   Scompliancei,Bcompliancei,VD,VBUILK,VSS,myNPLC,holdtime,
   sweepdelay,error,time,Id,Vg)
```

Section 3

ACS Python LPT library reference

In this section:

Introduction

For Python test modules (PTMs), ACS includes the ACS Python LPT library. The ACS Python LPT library contains functions that let you configure and control one or more instruments to perform parametric tests.

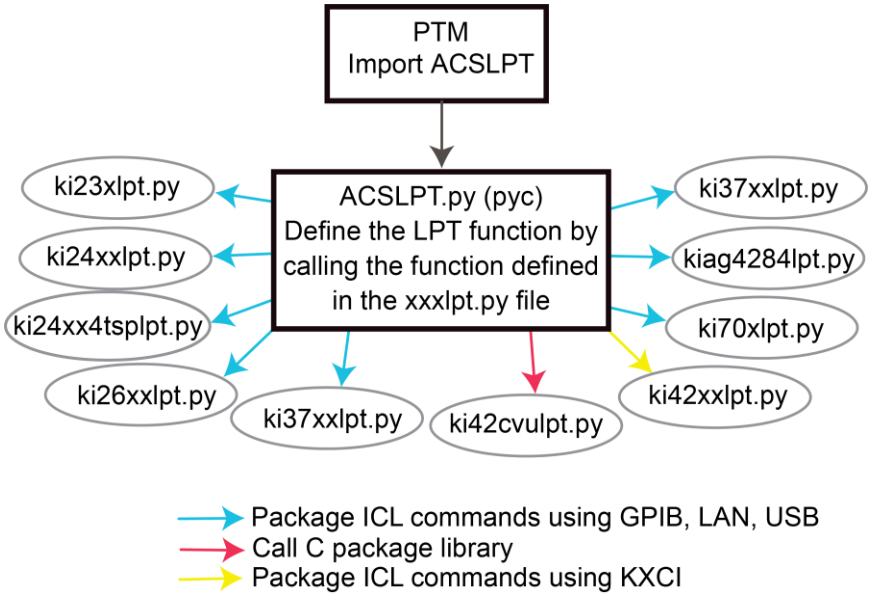
These commands are primarily in the same format as those in the Model 4200A-SCS library. The ACS Python LPT library is programmed in Python and can be used with Python test modules (PTMs). For more detail, refer to Creating PTM or STM test libraries and modules (on page 1-2).

You can import an ACS Python LPT command to control the following instruments:

- Series 23x instruments
- Series 2400 SourceMeter® instruments
- Series 2600B System SourceMeter® instruments
- Series 3700A System Switch instruments
- 4200A CVUs
- 4200/4210 SMUs
- 707B/708B series matrix systems
- Model 2010 Multimeters
- LCR 4284/4980 capacitance meters.

For more information, refer to the Python test module (PTM) configuration in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

Figure 23: ACS Python LPT call flow



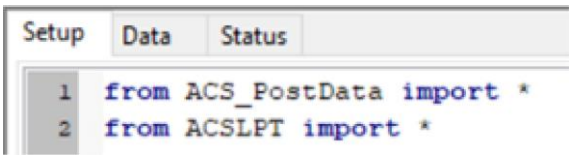
The following table describes how the CTM modules and ACS software functions interact.

ACS software compatibility		
ACS installed on:	**Interface:**	**Compatible library:**
Model 4200A-SCS	Normal (non-KXCI)	CTM functions
Model 4200A-SCS	KXCI and ethernet cable	Ki42cvulpt commands
Computer	KXCI and ethernet cable	Ki42cvulpt commands

Headers in Python test modules

When you create a new PTM in the test tree, headers are created (`ACS_PostData` and `ACSLPT`), as seen in the following figure. You can import a header after the PTM is created. For more information about importing a header, refer to Creating PTM or STM test libraries and modules (on page 1-2).

Figure 24: Import ACSLPT



NOTE

Before using the ACS post data and log commands, you need to import `ACS_PostData` to the header of a Python test module (PTM).

Python LPT functions

In the following tables, the LPT functions are grouped based on the instrument model. The functions for the instruments are listed alphabetically.

LPT functions		
Models 236, 237, 238 LPT functions		
devclr	devint	forcei
forcev	intgi	intgv
limiti	limitv	lorangei
lorangev	measi	measv
rangei	rangev	setauto
setmode	srangei	srangev

LPT functions		
Model 2010 Multimeter LPT functions		
devclr	devint	avgv
intgv	measv	rangev
setauto	setmode	getstatus

NOTE

The `lorangei` and `lorangev` functions for the Model 23x are equal to autorange regardless of the value of the parameter setting.

LPT functions				
Series 2400 SourceMeter® instruments LPT functions (SCPI and TSP mode)				
abort	adelay	asweepi	asweepv	bsweepi
bsweepv	asweepi	delay	devclr	forcev
intgi	intgv	limiti	limitv	measi
measv	rangei	rangev	rtfary	savgi
savgv	setauto	setmode	sintgi	sintgv
smeasi	smeasv	srangei	srangev	sweepi
sweepv	trigig	trigil	trigvg	trigvl

LPT functions				
Series 2600B SourceMeter® instruments LPT functions				
abort	adelay	asweepi	asweepv	avgi
avgv	bsweepi	bsweepv	devclr	devint
forcei	forcev	intgi	intgv	limiti
limitv	lorangei	lorangev	measi	measv
rangei	rangev	rtfary	savgi	savgv
setauto	setmode	sintgi	sintgv	smeasi
smeasv	srangei	srangev	sweepi	sweepv
trigig	trigil	trigvg	trigvl	

LPT functions			
Series 3700A System Switch LPT functions			
addcon	addpth	clrcon	conpin
conpth	delcon	delpth	devint

LPT functions			
Model 4200A-SCS LPT functions			
avgi	avgv	clrscn	clrtrg
delay	devclr	devint	disable
enable	execut	forcei	forcev
getinstatrr	getinstid	getstatus	imeast
Intgi	intgv	limiti	limitv
lorangei	lorangev	measi	measv
measz	rangei	rangev	rdelay
setauto	setfreq	setlevel	setmode
sweepi	sweepv	tstdsl	tstsel

LPT functions			
Models 707, 708 LPT functions			
addcon	addpth	clrcon	conpin
conpth	delcon	delpth	devint
insbind			

LPT functions		
Model 4200A CVU LPT functions		
devclr	devint	forcev
measz	rangei	setauto
Setfreq	setlevel	setmode

LPT functions			
Model 4284 LCR Meter LPT functions			
devclr	devint	forcev	getstatus
measz	rangei	setauto	setfreq
setlevel	setmode		

The following commands post data and information. They are general commands and are not specific to an instrument.

LPT functions		
ACS post data functions		
ACSPostArrayDouble	ACSPostArrayFloat	
ACSPostArrayInt	ACSPostDataDouble	
ACSPostDataFloat	ACSPostDataInt	
ACSPostData		

LPT functions		
Log functions		
logCritical	logError	logInfo

ACS Python LPT library commands

The following Python LPT library commands are listed in alphabetical order

abort

This command aborts the current source-delay-measure process.

Usage

```
abort
abort(*instName)
```

*instName	Optional; list of instruments to abort

Details

It is recommended that you call the abort function node in the user access point (UAP) only.

Example

```
#Abort all the instruments in the system.
abort()
#Abort SMU1 and CVU1.
abort(SMU1,CVU1)
```

Also see

None

addcon

This command adds a terminal-pin connection.

Usage

```
addcon(*instMTRX, ter, pin, *more_pin)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Indicates additional pins to connect

Details

Terminal and pin lists must have the same number of items. Terminals and pins are matched according to the sequence. If the numbers in the terminal and pin lists are not the same, the connections are performed based on the shortest list.

Normally, the `addcon` command supports the `ROW_COLUMN` mode of the matrix. When the matrix is set to `INSTRUMENT_CARD` mode, a row is assigned automatically to connect the terminal and the pin.

Example

```
addcon(MTRX1,SMU1,1)
addcon(SMU1,1)
addcon(SMU1H,1)
addcon(SMU1L,1)
addcon(SMU1,1,2,3)
addcon([SMU1,SMU2], [1,2])
addcon(SMU1, SMU2)
addcon(CVU1H,1)
```

Also see

clrcon (on page 3-13)
conpin (on page 3-15)
delcon (on page 3-18)

addpth

This command adds a terminal-pin connection through a row in a matrix.

Usage

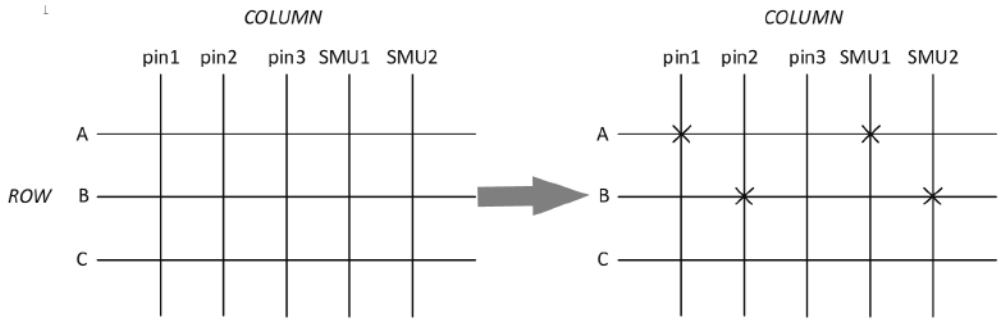
```
addpth(*instMTRX, ter, pin, row)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to be connected
pin	The list of pins to be connected
row	The row used to connect terminals and pins

Details

A terminal and a pin are connected through an assigned row to a matrix. In the following figure, terminal SMU1 and pin 1 are connected through Row A. Terminal SMU2 and pin 2 are connected through Row B. Note that you must use another instance of the `addpth` command to make a connection in another matrix.

Figure 25: addpth and conpth command example



This function can only be applied when the matrix is set to `INSTRUMENT_CARD` mode.

For more information about how to set the `INSTRUMENT_CARD` mode and `ROW_COLUMN` mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

NOTE

This TSP LPT command can only be used with a matrix connected using TSP-Link®.

Example 1

```
addpth(MTRX1,SMU1,1,'A')
addpth(SMU1,1,'A')
addpth(SMU1H,1,'A')
addpth(SMU1L,1,'A')
```

This is an example of using the `addpth` command with a Model 707/708 Switch Matrix.

Example 2

```
addpth(MTRX1,SMU1,1,'1')
addpth(SMU1,1,'1')
addpth(SMU1H,1,'1')
addpth(SMU1L,1,'1')
```

This is an example of using the `addpth` command with a Series 3700 System Switch.

Also see

clrcon (on page 3-13)
conpth (on page 2-7)
delpth (on page 3-18)

adelay

This command sets the source delay list for sweep functions using the commands `asweepi` or `asweepv`.

Usage

`adelay(`*`delayPoints`*`, `*`delayArray`*`)`

`delayPoints`	Number of delay points
`delayArray`	Delay array in seconds

Details

Use the `adelay` command to set the delay list for sweep functions with the force list `asweepi` or `asweepv` command. Each delay in the list is added to the step delay specified in `asweepi`/`asweepv` .

Note that the `adelay` command must be called before the `asweepi` or `asweepv` command.

Example

```
forceList = [0, 1, 1.5, 3, 3.2]

   #Set source delay list for a 5-point sweep.
adelay(5, [1e-4, 2e-4, 3e-4, 4e-4, 5e-4])
#Sweep measure current on SMU1.
smeasi(SMU1)
#Sweep source voltage according to the forceList with 5 steps and with 1e-3(s)
   stepdelay; based on the source delay list set by adelay, the actual delay list
   for the 5 steps is [1.1e-3, 1.2e-3, 1.3e-3, 1.4e-3, 1.5e-3](s).
swpData = asweepv(SMU1, 5, 1e-3, forceList)
devint()
#Post measured current data.
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#Post forced voltage value.
ACSPostArrayDouble("Vsource", swpData["rtfary"], 5)
```

Also see

None

asweepi/asweepv

This command performs a current or voltage sweep that forces a list of values and returns the measured data based on the measurement setup.

Usage

`asweepi(`*instName*`, `*stepNo*`, `*stepDelay*`, `*forceList*`)`
`asweepv(`*instName*`, `*stepNo*`, `*stepDelay*`, `*forceList*`)`

instName	Instrument name for forcing
stepNo	Step number of sweep
stepDelay	Delay before every sweep step
forceList	List of force values
return	*{"savgi_SMUX":[data_1, data_2,…, data_StepNo]}* *{"savgv_SMUX":[data_1, data_2,…, data_StepNo]}* *{"smeasi_SMUX":[data_1, data_2,…, data_StepNo]}* *{"smeasv_SMUX":[data_1, data_2,…, data_StepNo]}* *{"singti_SMUX":[data_1, data_2,…, data_StepNo]}* *{"singtv_SMUX":[data_1, data_2,…, data_StepNo]}* *{"rtfary":[value_1, value_2,…, value_StepNo]}*

Details

Use the `asweepi` command to sweep current or the `asweepv` command to sweep voltage using a list of forced values. This returns measured data based on the measurement setup for `savgi/savgv` or `sintgi`/`sintgv` or `smeasi`/`smeasv`. If the `rtfary` command is called, the return value includes programmed forced values of the sweep.

NOTE

The measurement setup function `savgi/savgv` or `sintgi/sintgv` or `smeasi/smeasv` must be called before `asweepi/asweepv`. The forced values return function `rtfary` must be called before `asweepi/asweepv`.

Example

```
forceList =  [0, 1, 1.5, 3, 3.2]

   #Sweep measure current on SMU1.
smeasi(SMU1)
#Return force value of sweep.
rtfary()
   #Sweep source voltage based on the forceList with 5 steps and with 0.001s delay
   swpData = asweepv(SMU1, 5, 0.001, forceList).
devint()

   #Post measured current data.
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#Post forced voltage value.
ACSPostArrayDouble("Vsource", swpData["rtfary"], 5)
```

Also see

savgi/savgv (on page 3-39)
sintgi/sintgv (on page 3-52)
rtfary (on page 3-38)
smeasi/smeasv (on page 3-53)

avgi/avgv

This command performs a series of measurements and averages the results.

Usage

`avgi(`*`unitname`*`, `*`iStepNo`*`, `*`dStepTime`*`)`
`avgv(`*`unitname`*`, `*`iStepNo`*`, `*`dStepTime`*`)`

unitname	The instrument identification code of the instrument; for example, `SMU1`
iStepNo	The number of steps averaged in the measurement; `1` to `160,000` (for 4200A it is `32767`)
dStepTime	The interval in seconds between each measurement; minimum practical time is approximately 0.0001 s (NPLC must be set at 0.001, for Model 4200A set at 2.5 µs)

Example

```
I1= avgi(SMU1, 100, 0.001)
```

Averages the results of 100 measurements from SMU1. Measurements are made at 1 ms intervals.

Also see

None

bsweepi/bsweepv

This command performs a current or voltage force sweep.

Usage

```
bsweepi(instName, startVal, endVal, stepNo, stepDelay, BIresult)
bsweepv(instName, startVal, endVal, stepNo, stepDelay, BVresult)
```

instName	Instrument name used to force sweep
startVal	Start value of the force sweep
endVal	End value of the force sweep
stepNo	Step number of the sweep
stepDelay	Delay before every sweep step
BIresult/BVresult	Empty list to store the force break value
return	{"savgi_SMUX":[data_1, data_2,…, data_StepNo]} {"savgv_SMUX":[data_1, data_2,…, data_StepNo]} {"smeasi_SMUX":[data_1, data_2,…, data_StepNo]} {"smeasv_SMUX":[data_1, data_2,…, data_StepNo]} {"singti_SMUX":[data_1, data_2,…, data_StepNo]} {"singtv_SMUX":[data_1, data_2,…, data_StepNo]} {"rtfary":[value_1, value_2,…, value_StepNo]}

Details

Use the `bsweepi` or `bsweepv` command to sweep the current or voltage using a defined start value, end value, step number, and step delay. The sweep breaks when the tested data exceeds the level set by the trigger functions `trigil`, `trigvl`, `trigig`, and `trigvg`. This returns measured data based on the measurement setup commands `savgi/savgv`, `sintgi/sintgv`, `smeasi`, or `smeasi/smeasv`, and the force value. If `rtfary` is called, the return value includes programmed force values of the sweep.

Note that the measurement setup function `savgi/savgv` or `sintgi/sintgv` or `smeasi/smeasv` must be called before `bsweepi/bsweepv`. The trigger functions `trigil/trigvl/trigig/trigvg` must be called before `bsweepi/bsweepv`. The force values return function `rtfary` must be called before `bsweepi/bsweepv`.

Example

```
#list to store force break value
BVresult = []
#set trigger of current low level
trigil(SMU1, 1e-11)
#measure current on SMU1
smeasi(SMU1)
#return force value of sweep
rtfary()
#sweep source voltage from 0~5V by 5 steps, with 0.001s delay,
#breaks according to the trigger level setting
swpData = bsweepv(SMU1, 0, 5, 5, 0.01, BVresult)
devint()
#post force break value
ACSPostArrayDouble("BV", BVresult[0])
#post measured current data
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#post forced voltage value
ACSPostArrayDouble("Vsource", swpData["rtfary"], 5)
```

Also see

rtfary (on page 3-38)
savgi/savgv (on page 3-39)
sintgi/sintgv (on page 3-52)
smeasi/smeasv (on page 3-53)
trigil/trigvl/trigig/trigvg (on page 3-56)

clrattrset

This command clears the present instrument setting in memory.

Usage

`clrattrset(*instName)`

`*instName`	Optional; list of instruments in the system to clear the memory

Details

Example

```
#clear the setting in memory for all the instruments in the system
clrattrset ()
#clear the setting in memory for SMU1 and SMU2
clrattrset(SMU1, SMU2)
```

Also see

None

clrcon

This command clears all the connections of all the matrices or specified matrices.

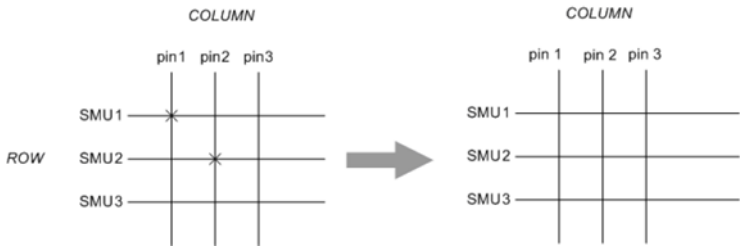
Usage

```
clrcon(unitname)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf, such as MTRX1

Details

Figure 26: clrcon (clear all) connections example



Example

```
clrcon()
clrcon(MTRX1)
```

Clear all matrix connections in the hardware configuration.
Clear only matrix1 connections.

Also see

None

clrscn

> **NOTE**
> Only for the Model 4200A-SCS.

This command clears the measurement scan tables associated with a sweep.

Usage

```
clrscn(*instName)
```

**instName*	List of instruments in the system performing the sweep (optional)

Details

The measurement-scan tables associated with a sweep are internal tables defined by `savgi/savgv`, `sintgi/sintgv`, `smeasi`, or `smeasi/smeasv`, or `rtfary`. The `clrscn` command is called after the sweep and it clears these tables.

Example

```
#measure forward
smeasi (SMU1)
forcur=sweep (SMU1, 0.0, 0.5, 10, 5.0e-3) #output 0 V to 0.5 V in 10 steps in
   5 ms steps
#clear the measurement scan tables associated with a sweep for all instruments in
   the system
clrscn()
#clear the measurement scan tables associated with a sweep for SMU1, SMU2
clrscn(SMU1, SMU2)
```

Also see

None

clrtrg

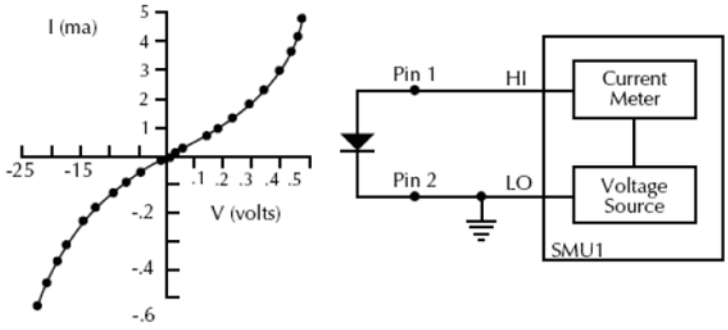
This command clears previously defined sweep trigger settings.

Usage

`clrtrg(*instName)`

`*instName`	Optional; list of instruments that have trigger settings cleared

Figure 27: clrtrg library command



Details

This command permits the use of `trigXl` or `trigXg` more than once at various levels within a single test sequence.

Example

```
#increase ramp to I = 5 mA
trigil(SMU1, 5.0e-3)
#measure forward
smeasi(SMU1)
#output 0 to 0.5V in 10 steps, each 5 ms duration
forcur=sweepv(SMU1, 0.0, 0.5, 10, 5.0e-3)
clrtrg() #clear trigger settings for all instruments in the system
#clear trigger settings only for SMU1
clrtrg(SMU1)
```

Also see

trigil/trigvl/trigig/trigvg (on page 3-56)

conpin

This command clears existing connections and adds new terminal-pin connections.

Usage

`conpin(*instMTRX, ter, pin, *more_pin)`

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Additional pins to connect

Details

Terminal and pin lists must have the same number of items and are arranged (by number) in the same sequence. If the numbers in the terminal and pin lists are not the same, the connections are performed on the shortest list and other entries are ignored.

Normally, the `conpin()` command supports the `ROW_COLUMN` mode of the matrix. When the matrix is set to `INSTRUMENT_CARD` mode, a row is assigned automatically to connect the terminals and the pins.

For more information about how to set the `INSTRUMENT_CARD` mode and `ROW_COLUMN` mode, refer to "Hardware configurations" in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

Example

```
conpin(MTRX1,SMU1,1)
conpin(SMU1,1)
conpin(SMU1H,1)
conpin(SMU1L,1)
conpin(SMU1,1,2,3)
conpin([SMU1,SMU2], [1,2])
conpin(CVU1H,1)
```

This example demonstrates different methods of using `conpin` to connect terminals to pins.

See also

addcon (on page 3-6)
clrcon (on page 3-13)
delcon (on page 3-18)

conpth

This command clears all matrix connections and makes a new terminal-pin connection through a row.

Usage

```
conpth(*instMTRX, ter, pin, row)
```

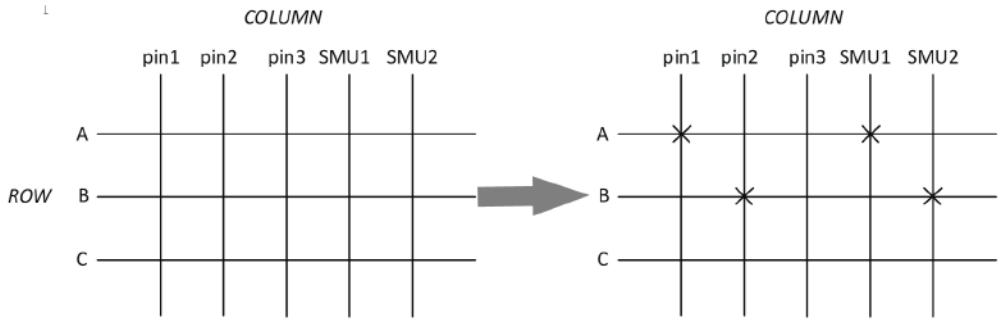
instMTRX	The matrix name in the hardware configuration (the name is optional)
ter	The list of terminals to connect
pin	The list of pins to connect
row	The row used to connect terminals and pins

Details

The terminal and pin are connected through an assigned row in a matrix. In the following figure, terminal SMU1 and pin1 are connected through row A. Terminal SMU2 and pin2 are connected through row B.

You must use one instance of the conpth command for each connection made in a matrix. You cannot use the conpth command to make a connection in multiple matrices (see the following figure).

Figure 28: addpth and conpth command example



You must use one instance of the *conpth* command for each connection made in a matrix. You cannot use the *conpth* command to make a connection in multiple matrices (see the following figure).

This function can only be applied when the matrix is set to INSTRUMENT_CARD mode.

Example 1

```
conpth(MTRX1,SMU1,1,'A')
conpth(SMU1,1,'A')
conpth(SMU1H,1,'A')
conpth(SMU1L,1,'A')
```

This is an example of using the `conpth` command with a Model 707B or 708B Switch Matrix.

Example 2

```
conpth(MTRX1,SMU1,1,'1')
conpth(SMU1,1,'1')
conpth(SMU1H,1,'1')
conpth(SMU1L,1,’1’)
```

This is an example of using the `conpth` command with a Series 3700A System Switch.

Also see

addpth (on page 3-7)
clrcon (on page 3-13)
delpth (on page 3-18)

delay

This command provides a user-programmable delay within a test sequence.

Usage

`delay(`*`iDelayTime`*`)`

iDelayTime	The time to delay in milliseconds

Example

```
#delay 0.001s
delay(0.001)
```

Also see

None

delcon

This command deletes a terminal-pin connection.

Usage

```
delcon(*instMTRX, ter, pin, *more_pin)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to connect
pin	The list of pins to connect
more_pin	Indicates additional pins to connect

Details

The terminal and pin must be connected before the command will delete the terminal pin connection, otherwise the connection is invalid.

Example

```
delcon(MTRX1,SMU1,1)
delcon(SMU1,1)
delcon(SMU1H,1)
delcon(SMU1L,1)
delcon(SMU1,1,2,3)
delcon([SMU1,SMU2], [1,2])
```

This example shows valid methods for defining the matrix, terminals, and pins.

Also see

addcon (on page 3-6)
clrcon (on page 3-13)
conpin (on page 3-15)

delpth

This command deletes a terminal-pin connection based on a specific row in a matrix.

Usage

```
delpth(*instMTRX, ter, pin, row)
```

instMTRX	Optional; the matrix name in the hardware configuration
ter	The list of terminals to disconnect
pin	The list of pins to be connected
row	The row used to connect terminals and pins

Details

The terminal, pin, and row must be in the same group when connected. Otherwise, the command is not valid.

Example 1

```
delpth(MTRX1,SMU1,1,'A')
delpth(MTRX1,1,'A')
delpth(SMU1H,1,'A')
delpth(SMU1L,1,'A')
```

This is an example of using the `delpth` command with a Model 707B or 708B Switch Matrix.

Example 2

```
delpth(MTRX1,SMU1,1,'1')
delpth(SMU1,1,'1')
delpth(SMU1H,1,'1')
delpth(SMU1L,1,'1')
```

This is an example of using the `delpth` command with a Series 3700A System Switch.

Also see

addpth (on page 3-7)
clrcon (on page 3-13)
conpth (on page 2-7)

devclr

This command sets all sources to a zero state.

Usage

`devclr(*instName)`

*`instName`	Optional; list of instruments and sources to set to a zero state

Details

This function sends output off commands or calls the Model 4200A `devclr` function. It does not work on a matrix.

If the system is configured using KCon, the Model 4200A `devclr` function executes. This function clears all sources sequentially.

Before clearing all Keithley Instruments supported instruments, GPIB-based instruments are cleared by sending all strings defined with `kibdefclr`. `Devclr` is implicitly called by `clrcon`, `devint`, `execut`, and `tstdsl`.

Example

```
#set all sources for all instruments in the system to zero state
devclr()
#set SMU1 to a zero state
devclr(SMU1)
#set all SMU1 and CVU1 sources to a zero state
devclr(SMU1, CVU1)
```

Also see

None

devint

This command resets the instruments and clears the system by opening all relays and disconnecting the pathways.

Usage

`devint(*instName)`

`*instName`	List of instruments that are reset and cleared (optional)

Details

This function resets all instrument meters and sources to their default state (refer to the specific instrument manual to find the default condition for each instrument).

The following tables indicate the default settings for the Series 2600B and the Model 2400 instruments after applying the `devint` command.

Default settings		
Series 2400 SourceMeter® instrument settings after `devint` is applied		
Output		Turns source off
Reset		Reset all bits of following registers to 0: ■ Standard event register ■ Operation event register ■ Measurement event register ■ Questionable event register
Long-form and short-form versions		Command word is sent in short-form version
ACS hardware configuration setting window	If interlock enabled	Enable interlock
	If rear panel enabled	Enable rear panel
	If beeper enabled	Disable beeper

Default settings	
Series 2600B SourceMeter® instrument settings after `devint` is applied	
Current range	0.1 A (all SMUs)
Output	Clear
Error queue	Clear
Status model	Reset all bits
Error display	Disable
PLC	0.001
	1 (for intgx)
Measure count	1
DTNS	Clear sweep table, clear trigger, clear garbage, then set all 26xx in DTNS group 0
Sense mode	Depends on ACS software preference setting
Reset	The default factory setting for each instrument

This command also performs the following actions before resetting the instruments:

- Clear all sources by calling `devclr`
- Clear the matrix cross-points by calling `clrcon`
- Clear the trigger tables by calling `clrtrg`
- Clear the sweep table by calling `clrscn`
- Reset the GPIB instruments by sending the string defined with `kibdefint`
- Stop the pulse generator card and select the standard pulse mode and its default settings (for instance, `*RST`)

Note that `devint` is implicitly called by `execut` and `tstdsl`.

Example

```
#reset and clear the instruments in the system
devint()
#reset and clear SMU1
devint(SMU1)
#reset and clear SMU1 and MTRX1
devint() (SMU1, MTRX1)
```

Also see

None

disable

This command stops the timer, sets the time value to zero, and stops the timer reading.

Usage

`disable(`*`timerName`*`)`

`timerName`	Timer name defined by the `enable()` command

Example

```
disable('TIMER1')
```

Stops TIMER1 and sets the time to zero.

Also see

enable (on page 3-22)
imeast (on page 3-31)

enable

This command enables a timer, which can be used with the command `imeast()` to measure time.

Usage

`enable('`*`timerName`*`')`

`timerName`	The name of the timer to be enabled

Example

```
enable('TIMER1')
```

Also see

disable (on page 3-22)
imeast (on page 3-31)

execut

This command prompts the system to wait for the previous test sequence to execute.

Usage

```
execut(*instName)
```

*instName	Optional; list of instruments waiting for the test sequence to execute

Details

For the Model 4200A or Series 2600B SourceMeter® instruments, this function waits for all previous LPT commands to complete and then issues a `devint` command.

Example

```
#execute all the instruments in the system
execut()
#execute SMU1
execut(SMU1)
```

Also see

None

forcei/forcev

This command programs a sourcing instrument to generate a voltage or current at a specific level.

Usage

```
forcei(unitname, dValue)
forcev(unitname, dValue)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
dValue	The level of the voltage or current forced in volts or amperes

Details

Example

```
#force current 1e-6 A on SMU1
forcei(SMU1, 1e-6)
#force voltage 0.02 V on SMU2
forcev(SMU2, 0.02)
```

Sets up SMU1 to source 1 µA and SMU2 to source 0.02 V.

Also see

None

getcommon

This command gets common attributes from the global dictionary.

Usage

```
getcommon()
```

Details

Return key list:

- **UNITLIST**: List of instrument IDs in the system
- **PLC**: Power line cycle
- **Pin**: Information about the pin

Example

```
print getcommon()
{'PLC': '60HZ', 'UNITLIST': ['GNDU', 'PRBR1', 'SMU1', 'TIMER1']}
```

Also see

None

getinstattr

This command gets the instrument attribute from the instrument name string.

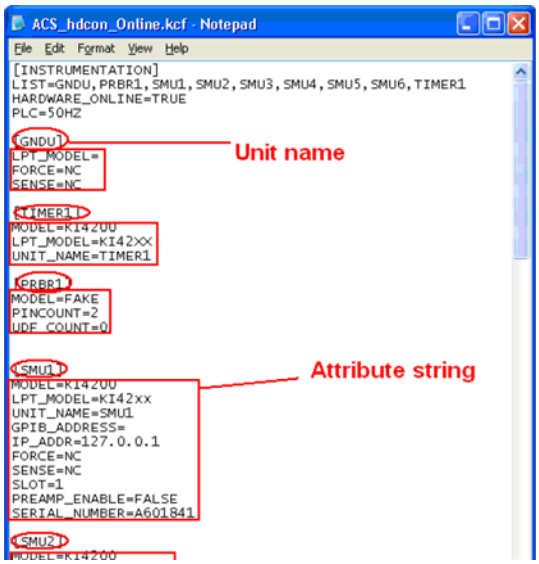
Usage

```
getinstattr(unitname, attr_str)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
attr_str	The attribute string list is in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf

Details

Figure 29: Unit name and Attribute string .kcf file



Return value: `INVAL_INST_ID`

`-1` ---- This function does not apply to this unit.

Example

```
getinstattr(SMU1, "GPIB_ADDRESS")
print getinstattr(SMU1, "MODEL")
```

An example return from this example is:
`KI4200A`

Also see

None

getinstid

> **NOTE**
> Only for the Model 4200A-SCS.

This command gets the instrument identifier (ID) from the instrument name string.

Usage

`getinstid(`*`unitname`*`)`

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`

Example

```
print getinstid(SMU1)
```

An example return is:

```
4100
```

Also see

None

getstatus

> **NOTE**
> Only for the Model 4200A-SCS.

This command returns the operating state of the specified instrument.

Usage

`getstatus(`*`unitname`*`, `*`iCode`*`)`

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`
iCode	The parameter of the query

Details

Return value: Data returned from the instrument. `getstatus` returns one item

Valid Errors: `UT_INVLDPRM` invalid parameter error; status item parameter is not supported for this device; requested status code is invalid for selected device

A list of supported `getstatus` query parameters for a SMU are provided in the following table.

Getstatus parameters		
`iCode`	**Comment**	
`KI_IPVALUE`	The presently programmed output value.	Current value (I output value).
`KI_VPVALUE`		Voltage value (V output value).
`KI_IPRANGE`	The presently programmed range.	Current range (full-scale range value or 0.0 for autorange).
`KI_VPRANGE`		Voltage range (full-scale range value or 0.0 for autorange).
`KI_IARANGE`	The presently active range.	Current range (full-scale range value).
`KI_VARANGE`		Voltage range (full-scale range value).
`KI_IMRANGE`	The range used when the last measurement was performed.	For autorange, the range at which the previous I measurement was performed.
`KI_VMRANGE`		For autorange, the range at which the previous V measurement was performed.
`KI_COMPLNC`	Active compliance status.	Bitmapped values: 2 = LIMIT (at the compliance limit set by `limit`*`X`*). 4 = RANGE (at the top of the range set by `range`*`X`*).
`KI_RANGE_COMPLIANCE`	Active compliance status for fixed range.	In range compliance if 1.
`KI_COMPLNC_EVER`	Compliance history.	Reset by reading compliance history and by `devint`.

ki20xxlpt Getstatus parameters	
iCode	**Comment**
`KI_VPRANGE`	The presently programmed voltage range.
`KI_VARANGE`	The presently active voltage range.
`KI_VMRANGE`	The range used when the last measurement was made. For autorange, the range at which the previous V measurement was made.

Example

```
gstatus=getstatus(SMU1, KI_COMPLNC)
```

Returns the state of compliance for SMU1.

Also see

None

gpibenter

This command is used to read a device-dependent string from an instrument connected to a GPIB, LAN, or USB interface.

Usage

```
gpibenter(unitname, max_size)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
max_size	A value specifying the maximum number of characters you want to receive; maximum length can be a number from 0 to 32767 (hex FFFF)

Details

The return value is (tupple type) (receive str, length, status).

If the string is read back successfully, the output in the following table is displayed.

Output data display	Data/reason
receive str	The received string.
length	The length of the received string.
status	0

If the reading times out the output in the following table is displayed.

Output data display	Data/reason
receive str	String of TIMEOUT.
length	The length of TIMEOUT string.
status	8

Example

```
rvalue = gpibenter(SMU2, 100)
```

Also see

None

gpibsend

This command sends a device-dependent command to an instrument connected to GPIB, LAN, or USB.

Usage

`gpibsend(`*`unitname`*`, `*`cmd_string`*`)`

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`
cmd_string	A command string sent to the device; note that the terminating characters are automatically added to the end of this string when it is sent; the default terminator is a line feed character

Details

Return value: A variable which indicates the success or failure of the data transfer.

Example

```
gpibsend(SMU1, 'devint()')
gpibsend(GPI1, "L2X")
```

Also see

None

gpibspl

This command conducts a serial poll that reads the status of an instrument connected to the GPIB interface.

Usage

`gpibspl(`*`unitname`*`)`

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`

Details

The following indicates the name of the output data string displayed in the data panel.

Return value: (tuple type) − (receive number, status) or error code

If the serial poll is read back successfully, the output in the following table is displayed.

Output data display	Data/reason
Receive number	Serial poll number received
Status	0

If the reading times out, the output in the following table is displayed.

Output data display	Data/reason
Receive number	Predefined string indicating TIMEOUT
Status	8

Example

```
poll1 = gpibspl(SMU1)
```

Also see

None

insbind

This command binds the high-voltage (HV) SMU and high-voltage ground (HVGND) to the CVU HI, LO, and ground (CVU_HI/LO/GRD) to the bias tee through pins 09, 10, and 11 (PIN09/10/11).

Usage

`insbind(`*`ter1`*`, `*`ter2`*`)`

ter1	CVU terminal to be connected
ter2	SMU terminal to be connected

Details

The pin number is defined in the `ACS_setting.ini` file. Only the terminals related to HV SMU, HVGND, and HV CVU are supported as inputs.

Example

```
insbind(CVU1H, SMU1)
insbind(CVU1G, HVGND)
```

Also see

None

imeast

This command forces a read of the timer and returns the result.

Usage

```
imeast(unitname)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf, such as TIMER1 or TIMER2

Details

This command applies to all timers. You must call enable (TIMERn) first. The return value is the elapsed time from enable (TIMER1).

Example

```
t1 = imeast(TIMER1)
```

Also see

enable (on page 3-22)
disable (on page 3-22)

intgi/intgv

This command performs voltage or current measurements averaged over a user-defined period.

Usage

```
intgi(unitname)
intgv(unitname)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf

Details

This averaging is done in the hardware through the integration of the analog measurement signal over a period of specified time (usually this is completed during one AC-line cycle). The integration is automatically corrected for 50 Hz or 60 Hz power mains.

Example

```
i1= intgi(SMU1)
```

Also see

None

limiti/limitv

This command allows you to specify a current or voltage limit.

Usage

```
limiti(unitname, dValue)
limitv(unitname, dValue)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
dValue	The maximum level of the current or voltage; refer to **Details**

Details

Use `limiti` to limit the current of a voltage source. Use `limitv` to limit the voltage of a current source.

The limit value is bidirectional. For example, `limitv("SMU1", 10.0)` limits the voltage of the current source of SMU1 to ±10.0 V. The example `limiti("SMU1", 1.5E-3)` limits the current of the voltage source of SMU1 to ±1.5 mA.

Example

```
#set the current limit of SMU1 to ±0.01 A
limiti(SMU1, 0.01)
#set the voltage limit of SMU2 to ±200 V
limitv(SMU2, 200)
```

Set the current limit for SMU1 to ±0.01 A and the voltage limit of SMU2 to ±200 V.

Also see

None

lorangei/lorangev

This command defines the lowest autorange limit.

Usage

```
lorangei(unitname, dValue)
lorangev(unitname, dValue)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
dValue	The instrument range in voltage or amperes

Details

The `lorange` command is used with autoranging to limit the number of range changes. Limiting the range changes saves test time.

For the Model 4200A-SCS, if the instrument was on a range lower than the one specified by `lorange`, the range is changed. The 4200A-SCS automatically provides any range change settling delay that may be necessary due to this potential range change. Once defined, `lorange` is in effect until a `devclr`, `devint`, `execut`, or another `lorange`*`X`* executes.

If you are using this command with a Keithley Model 23x instrument, it will autorange. The second *`dValue`* is ignored.

This command cannot be used with the Series 2400 SourceMeter® instrument.

Example

```
#set lowest autorange limit of current on SMU1 to 1E-6 A
lorangei(SMU1, 1E-6)
#set lowest autorange limit of voltage on SMU2 to 0.2 V
lorangev(SMU2, 0.2)
```

This example sets the low autorange current limit to on SMU1 to 1 µA and the low autorange voltage limit on SMU2 to 0.2 V.

Also see

None

measi/measv

This command allows the measurement of current or voltage.

Usage

```
measi(unitname)
measv(unitname)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf

Details

For this measurement, the signal is sampled for a specific period. This sampling time for the measurement is called the integration time.

For the `measX` function, the integration time is fixed at 0.01 PLC. For 60 Hz line power, 0.01 PLC = 166.67 µs (0.01 PLC/60 Hz).

For 50 Hz line power, 0.01 PLC = 200 µs (0.01 PLC/50 Hz).

Example

```
i1= measi(SMU1)
```

This is a variable name presenting measure current

Also see

None

measz

This command performs an impedance measurement on a CVU or another capacitance measuring instrument.

Usage

```
measz(unitname, iModel, iSpeed)
[result1, result2] = measz()
```

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`
iModel	Measurement model; see **Details**
iSpeed	Measure speed: `KI_CVU_SPEED_FAST`, `KI_CVU_SPEED_NORMAL`, or `KI_CVU_SPEED_QUIET`
result1	Measurement model; see **Details**
result2	Measure speed: ■ `KI_CVU_SPEED_FAST` ■ `KI_CVU_SPEED_NORMAL` ■ `KI_CVU_SPEED_QUIET`

Details

The measurement models are listed in the following table.

The impedance measurement setting for a CVU is `KI_CVU_SPEED_FAST` that performs a fast measurement with higher noise.

Measurement settings			
unitname (Model name)	**iModel (Measurement Model)**		**Parameter values**
CVU1	ZTH	Impedance (Z) and phase (in radians)	KI_CVU_TYPE_ZTH or 0
	RjX	Resistance and reactance	KI_CVU_TYPE_RJX or 1
	CpGp	Parallel capacitance and conductance	KI_CVU_TYPE_CPGP or 2
	CsRs	Series capacitance and resistance	KI_CVU_TYPE_CSRS or 3
	CpD	Parallel capacitance and dissipation factor	KI_CVU_TYPE_CPD or 4
	CsD	Series capacitance and dissipation factor	KI_CVU_TYPE_CSD or 5
	RAW	Raw data from measure	KI_CVU_TYPE_RAW or 6
CMTR1	Z-thr	Impedance (Z) and phase (in radians)	KI_AGCV_TYPE_CPD or 0
	R-X	Resistance and reactance	KI_AGCV_TYPE_RX or 1
	Cp-G	Parallel capacitance and equivalent parallel conductance	KI_AGCV_TYPE_CPG
	Cs-Rs	Series capacitance and resistance	KI_AGCV_TYPE_CSRS
	Cp-D	Parallel capacitance and dissipation factor	KI_AGCV_TYPE_CPD
	Cs-D	Series capacitance and dissipation factor	KI_AGCV_TYPE_CSD

Measurement settings			
unitname (Model name)	**iModel (Measurement Model)**		**Parameter values**
	Cp-Q	Parallel capacitance and quality factor (inverse of D)	KI_AGCV_TYPE_CPQ
	Cs-Q	Series capacitance and quality factor (inverse of D)	KI_AGCV_TYPE_CSQ
	Lp-D	Inductance value measured with parallel-equivalent circuit model and dissipation factor	KI_AGCV_TYPE_LPD
	Lp-Q	Inductance value measured with parallel-equivalent circuit model and quality factor (inverse of D)	KI_AGCV_TYPE_LPQ
	Lp-G	Parallel inductance value and equivalent parallel conductance	KI_AGCV_TYPE_LPG
	Lp-Rp	Parallel inductance value and equivalent parallel resistance	KI_AGCV_TYPE_LPRP
	Ls-D	Series inductance value and dissipation factor	KI_AGCV_TYPE_LSD
	Ls-Q	Series inductance value and quality factor (inverse of D)	KI_AGCV_TYPE_LSQ
	Ls-Rs	Series inductance value and equivalent resistance	KI_AGCV_TYPE_LSRS
	Z-thd	Impedance (Z) and phase (in degrees)	KI_AGCV_TYPE_ZTD
	Cp-Rp	Parallel capacitance and equivalent resistance	KI_AGCV_TYPE_CPRP
	G-B	Equivalent parallel conductance and susceptance	KI_AGCV_TYPE_GB
	Y-thd	Admittance and phase (in degrees)	KI_AGCV_TYPE_YTD
	Y-thr	Admittance and phase (in radians)	KI_AGCV_TYPE_YTR
	Vdc-Idc	Direct-current voltage and direct-current electricity	KI_AGCV_TYPE_VDID

Example

```
measData = measz(CVU1, KI_CVU_TYPE_CSRS, KI_CVU_SPEED_NORMAL)
```

Also see

None

rangei/rangev

This command selects a measurement range and prevents the selected instrument from autoranging.

Usage

```
rangei(unitname_str, dvalue)
rangev(unitname_str, dvalue)
```

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf
dValue	The value of the highest measurement made; see **Details**

Details

By selecting a range, the time required for autoranging is eliminated.

The most appropriate range for this measurement is selected automatically. If the range is set to `0`, the instrument uses autorange unless you have a Series 2600B SourceMeter® instrument.

Example

```
#Select current range of 2 mA.
rangei(SMU1, 2.0E-3)
#Select voltage range of 20 V.
rangev(SMU2, 20)
```

Sets the current measurement range to 2 mA and the voltage measurement range to 20 V.

Also see

None

rdelay

This command provides a user-programmable delay in a test sequence.

Usage

```
rdelay(dDelayTime)
```

dDelayTime	The delay duration in seconds

Details

This command delays the test sequence by a specified number of seconds.

Example

```
--delay 0.001 s
rdelay(0.001)
```

Add a 1 ms delay.

Also see

None

rtfary

This command returns the programmed force values of the sweep.

Usage

```
rtfary()
```

Details

If `rtfary` is called before the sweep functions `sweepi`, `sweepv`, `asweepi`, `asweepv`, `bsweepi`, or `bsweepv`, the return value includes the programmed force values of the sweep in the following format:

```
{"rtfary":[value_1, value_2,…, value_StepNo], …}
```

The number of values returned by the `rtfary` command is determined by the number of force points generated by the sweep.

Example

```
#sweep measure current on SMU1
smeasi(SMU1)
#return force value of sweep
rtfary()
#sweep source voltage from 0 V to 5 V by 5 steps with a 0.001 s delay
swpData = sweepv(SMU1, 0, 5, 5, 0.001) devint()
#post measured current data
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#post forced voltage value
ACSPostArrayDouble("Vsource", swpData["rtfary"], 5)
```

Also see

asweepi/asweepv (on page 3-54)
bsweepi/bsweepv (on page 3-11)
sweepi/sweepv (on page 3-54)

savgi/savgv

This command determines the measurement method to average the measurement for the current or voltage sweep.

Usage

```
savgi(instName, avgNum, delayTime)
savgv(instName, avgNum, delayTime)
```

instName	Instrument name for the measurement
avgNum	Average number
delayTime	Delay time

Details

Use the `savgi` or `savgv` command to set the measurement method of the current or voltage sweep. Note that the measurement setup function `savgi`/`savgv` must be called before the sweep functions `sweepi`/`sweepv`, `asweepi`/`asweepv`, or `bsweepi`/`bsweepv`.

Example

```
#sweep measure current on SMU1
savgi(SMU1)
#sweep source voltage from 0~5V by 5 steps and with 0.001s delay
swpData = sweepv(SMU1, 0, 5, 5,0.001)

devint()
#post measured current data
ACSPostArrayDouble("Imeas", swpData["savgi_SMU1"], 5)
```

Also see

asweepi/asweepv (on page 3-9)
bsweepi/bsweepv (on page 3-11)
sweepi/sweepv (on page 3-54)

setauto

This command enables autoranging and cancels any previous `rangeX` command for the specified instrument.

Usage

`setauto(unitname)`

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf

Details

When an instrument is returned to the autorange mode, it remains in its present range for measurement purposes. The source range changes immediately.

Due to the dual mode operation of the SMU (voltage versus current), `setauto` places both voltage and current ranges in autorange mode.

Example

```
#enable autorange mode
setauto(SMU1)
```

Sets SMU1 to use autorange for the voltage and current ranges.

Also see

None

setfreq

This command provides a capacitance voltage (CV) test and sets the frequency for the AC drive.

Usage

`setfreq(unitname, dFreq)`

unitname	The instrument name in the directory \\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf; only CVUn and CMRTn are supported
dFreq	Frequency of the AC drive in hertz

Details

This is a CVUn and CMTRn test command.

Example

```
#set the frequency to 1e6Hz
setfreq(CVU, 1e6)
```

Also see

None

setlevel

This command provides a capacitance voltage (CV) test and sets the voltage level of the AC drive.

Usage

`setlevel(`*`unitname`*`, `*`dSignalLevel`*`)`

unitname	The instrument name in the directory `\\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`; only CVUn and CMTRn are supported
dSignalLevel	The voltage level of the AC drive

Details

This is a CVUn and CMTRn test command. There are different voltage level ranges for the CVU and CMTR, which can be found in the 4200A-SCS datasheet (https://www.tek.com/en/datasheet/4200a-scs-parameter-analyzer).

Example

```
status = setlevel(CVU1, 0.05)
```

Also see

None

setmode

This command sets the instrument-specific operating mode parameters.

Usage

`setmode(`*`unitname`*`, `*`iModifier`*`, `*`dValue`*`)`

unitname	The instrument identification code of the instrument; for example, `SMU1`
iModifier	The `setmode` modifier; select the links below for specific parameters based on the model
dValue	The value of the `setmode` modifier

Details

The `setmode` command allows control over certain instrument-specific operating characteristics. Refer to the documentation for your instruments to determine what each instrument supports.

The following topics, as indicated in *Also see* below, contain the `setmode` modifiers arranged by the instrument model number.

Example

```
status = setmode("CVU1", KI_CVU_OPEN_COMPENSATE, isCmpstOpen=0)
```

This example disables compensation constants for open load and short on CVU1.

Also see

Model 20xx LPT parameters (on page 3-42)
Model 23x setmode modifiers (on page 3-43)
Series 2400 setmode modifiers (on page 3-45)
Series 2600B setmode modifiers (on page 3-45)
Model 4200A setmode modifiers (on page 3-46)
Model 4200A CVU setmode modifiers (on page 3-49)
Model 4284 setmode modifiers (on page 3-51)

Model 20xx LPT parameters

Model 20xx LPT parameters			
Unit name	**iModifier**	**dValue**	**Comments**
VMTR1	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time to use for the intgv command.
	KI_AVGMODE	KI_MEASX KI_INTEGRATE	Controls what kind of readings are made for avgv calls.

Model 23x setmode modifiers

Model 23x LPT parameters			Comments
Unit name	**iModifier**	**dValue**	
SMU1	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time the SMU uses for the intgx command. The default devint value is 1.0 PLC. The valid range is 0.001 PLC to 25.0 PLC.
	KI_SENSE	KI_SENSE_LOCA (or 0)	Sets local sense mode (2-wire).
		KI_SENSE_REMO (or 1)	Sets remote sense mode (4-wire).
	KI_TRIG_IN	KI_TRIG_IN_CONT = 0	Input triggers control when source, delay, and measure operations occur. This setting continuously processes all source-delay-measure (SDM) cycles.
		KI_TRIG_IN_SRC = 1	Each trigger processes an SDM cycle.
		KI_TRIG_IN_DLY = 2	Initial trigger sets the source. Each subsequent trigger initiates a delay and measure, then sets the source of the next SDM cycle.
		KI_TRIG_IN_SRCDLY = 3	Two-trigger process for each SDM cycle. First trigger sets the source. Second trigger initiates a delay and then measures.
		KI_TRIG_IN_MSR = 4	Initial trigger sets the source and initiates a delay. Second trigger initiates a measurement, then sets the source and initiates a delay for the next SDM cycle.
		KI_TRIG_IN_SRCMSR = 5	Two triggers process each SDM cycle. First trigger sets the source and initiates a delay. Second trigger initiates a measurement.
		KI_TRIG_IN_DLYMSR = 6	Initial trigger sets the source. Two triggers process each SDM cycle. First trigger initiates a delay. Second trigger initiates a measurement and sets the source of next SDM cycle.
		KI_TRIG_IN_SRCDLYMSR = 7	Three triggers process each SDM cycle. First trigger sets the source. Second trigger initiates a delay. Third trigger initiates a measurement.

Model 23x LPT parameters			Comments
Unit name	**iModifier**	**dValue**	
		KI_TRIG_IN_PULSE = 8	Pulse-sweep trigger. Each trigger processes the on time and off time of each pulse in the sweep. Two measurements are made on each pulse.
	KI_TRIG_SOURCE	KI_TRIG_X = 0	Input trigger origin. The input trigger stimulus may be provided by the front-panel manual trigger function, an external device that applies a TTL-level pulse to the TRIGGER connector on the rear panel, or an appropriate IEEE-488 operation. IEEE X origin; X sent over IEEE-488 bus.
		KI_TRIG_GET = 1	Group execute trigger.
		KI_TRIG_TALK = 2	Unit addressed to talk over IEEE-488 bus.
		KI_TRIG_EXTERNAL = 3	Negative-going TTL-level pulse applied to TRIGGER connector.
		KI_TRIG_INTERNAL = 4	Front-panel MANUAL trigger function or HO command over IEEE-488 bus.
	KI_TRIG_OUT	KI_TRIG_OUT_NONE = 0	Output trigger generation: No output triggers.
		KI_TRIG_OUT_SRC = 1	Outputs a trigger pulse after every source phase.
		KI_TRIG_OUT_DLY = 2	Outputs a trigger pulse after every delay phase.
		KI_TRIG_OUT_SRCDLY = 3	Outputs a trigger pulse after every source phase and delay phase.
		KI_TRIG_OUT_MSR = 4	Outputs a trigger pulse after every source phase and measure phase.
		KI_TRIG_OUT_SRCMSR = 5	Outputs a trigger pulse after every source phase and measure phase.
		KI_TRIG_OUT_DLYMSR = 6	Outputs a trigger pulse after every delay phase and measure phase.
		KI_TRIG_OUT_SRCDLYMSR = 7	Outputs a trigger pulse after every source phase, delay phase, and measure phase.
		KI_TRIG_OUT_PULSE = 8	For pulse sweeps; outputs a trigger pulse after the end of each off-time measure.
	KI_SWEEPEND_TRIGOUT	KI_SWEEPEND_TRIGOUT_EN = 1	When enabled, an output trigger pulse occurs at the end of the sweep.
		KI_SWEEPEND_TRIGOUT_DIS = 0	
	KI_AVGNUMBER	0, 2, 4, 8, 16, 32	Number of readings to use to make an average; 0 means disable average filter.

Series 2400 setmode modifiers

Series 2400 instruments LPT parameters			Comments
Unit name	**Modifier**	**Value**	
SMU1	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time the SMU uses for the intgx command. The default devint value is 1.0 PLC. The valid range is 0.01 PLC to ~10 PLC (DC) and 0.004 PLC to ~0.1 PLC (2430 pulse mode).
SMU1 (only 2430 SMU)	KI_TRIG_IN_CONT	<value>	Sets the output pulse count.
SMU1	PULSE_MODE_PULSE	VOLT: Voltage source CURR: Current source	Selects the pulse mode and pulse source function.
	PULSE_MODE_WID	<value>	Selects the pulse mode and sets the pulse width.
	PULSE_MODE_DELAY	<value>	Selects the pulse mode and sets pulse delay.

Series 2600B setmode modifiers

Series 2600B instruments LPT parameters			Comments
Unit name	**Modifier**	**Value**	
SMU1	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time the SMU uses for the intgx command. The default devint value is 1.0 PLC. The valid range is 0.001 PLC to 25.0 PLC.
	KI_AVGMODE	KI_MEASX	Controls what kind of readings are taken for avg*X* calls; KI_MEASX is the devint default value.
		KI_INTEGRATE	Uses the time specified by the setmode command.
	KI_OFFMODE	KI_OFF_NORM	Sets source output-off mode to output 0 V when the output is turned off.
		KI_OFF_ZERO	Resets the output to 0 (either voltage or current) when output is off.
		KI_OFF_OPEN	Opens the output relay when the output is turned off.
	KI_SENSE	KI_SENSE_LOCA	Sets local sense mode (2-wire).
		KI_SENSE_REMO	Sets remote sense mode (4-wire).
		KI_SENSE_CALA	Sets calibration sense mode.

Model 4200A setmode modifiers

Support	LPT Parameters			Comments
	Instrument ID	**Modifier**	**Value**	
Supported	KI_SYSTEM	KI_TRIGMODE	KI_MEASX KI_INTEGRATE KI_AVERAGE KI_ABSOLUTE KI_NORMAL	Redefines all existing triggers to use a new method of measurement.
		KI_AVGNUMBER	*value*	Sets the number of readings to make when KI_TRIGMODE is set to KI_AVERAGE.
		KI_AVGTIME	*value* (in seconds)	Sets the time between readings when KI_TRIGMODE is set to KI_AVERAGE.
No operations performed*	KI_SYSTEM	KI_MX_DEFMODE	KI_HIGH KI_LOW	Sets the default mode to high-current mode or low-current mode. This setting remains in effect until the end of the present session and is not reset by the devint command.
		KI_HICURRENT	KI_ON	Forces the matrix into high-current mode. The mode reverts to the default at the next devint call unless the configuration file sets this parameter to reset on a clrcon call.
		KI_CC_AUTO	KI_ON KI_OFF	Turns automatic compliance clear processing on or off; the devint command resets this value to KI_ON.
		KI_CC_SRC_DLY	*value*	The minimum time after a source value change before a compliance clear scan may start. This represents the time that it takes the circuit under test to settle after a source value change and prevents false compliance detection due to transients.
		KI_CC_COMP_DLY	*value*>	The time between compliance scans while processing the compclr command. This also represents the time that it takes the circuit under test to settle after a source value change and prevents false compliance detection due to transients. However, the source value changes are only due to removing the instrument from an artificial compliance state.
		KI_CC_MEAS_DLY	*value*	The minimum time after the last source value change before a measurement can be made. This represents the time it takes the circuit under test to settle to the requested level for the subsequent measurements.

Support	LPT Parameters			Comments
	Instrument ID	Modifier	Value	
Supported	SMU*n*	KI_INTGPLC	*value* (in numbers of line cycles)	Specifies the integration time the SMU uses for the intgx and sintgx commands. The default devint value is 1.0 PLC. The valid range is 0.01 PLC to 10.0 PLC.
		KI_AVGMODE	KI_MEASX KI_INTEGRATE	Controls the type of readings that are made for avg*X* calls. The devint default value is KI_MEASX. When KI_INTEGRATE is specified, the integration time specified by the KI_INTGPLC setmode call is used.
No operations performed*	SMU*n*	KI_IMTR		Sets up the SMU as a current meter. The ranges used are representative of the type of instrument being simulated. **NOTE** This setmode turns the source on.
			KI_S400	Sets the SMU to use ranges equivalent to the Model S400.
			KI_DMM	Sets the SMU to use ranges equivalent to a DMM (lowest range is 100 µa). Provides a lower resolution, fast measurement. Used for high-current applications.
			KI_ELECTROMETER	Sets the SMU to use ranges equivalent to an electrometer. Provides best measurement resolution, but has a slower measurement time. Used for low-current measurements.
		KI_LIM_INDCTR	Any	Controls that measured value that is returned if the SMU is at its programmed limit. The devint default is SOURCE_LIMIT (7.0e22). **NOTE** The SMU always returns INST_OVERRANGE (1.0e22) if it is on a fixed range that is too low for the measured signal.
		KI_LIM_MODE	KI_INDICATOR KI_VALUE	Controls whether the SMU will return an indicator value when in limit or overrange, or if the actual value is measured. The default mode after a devint is to return an indicator value.
		KI_RANGE_DELAY	<value> (in seconds) ranges from -2147493.647 seconds to +2147483.647 seconds	Specifies an additional delay time for the SMU driver to add to the range settle delay time whenever it is changing a preamplifier range. Value may be negative to shorten the overall range change delay. In no event is the overall delay time less than the preamplifier circuit hardware switching time. The devint default value is 0.0.

Support	LPT Parameters			Comments
	Instrument ID	**Modifier**	**Value**	
		KI_RANGE_SETTLE	0.01 0.1 1.0 2.5 5.0 10.0	Controls how long the SMU driver delays when changing a preamplifier range. Value is specified in percent settling accuracy (using the listed valid percent values). The actual delay time depends on which range the preamp is switched from and the range it is switched to. The devint fault value is 1.00.
		KI_VMTR		Sets up the SMU as a voltmeter. The ranges used are representative of the type of instrument being simulated. **NOTE** This setmode turns the source on.
			KI_S400	Sets the SMU to use ranges equivalent to the Model S400.
			KI_DMM	Sets the SMU to use range equivalent to a DMM. Provides a low impedance, fast measurement. Used for low-voltage applications.
			KI_ELECTROMETER	Sets the SMU to use ranges equivalent to an electrometer. Provides a high input impedance, but has a slower measurement time. Used for high-resistance measurements.
* These modifiers do not perform any operations in the Model 4200A-SCS. They are included for compatibility reference for existing S600 programs that use the setmode function. S600 programs can be ported to the Model 4200A-SCS.				

Model 4200A CVU setmode modifiers

Model 4200A CVU LPT parameters			Comments
Unit name	**Modifier**	**Value**	
CVU1	KI_CVU_CABLE_CORRECT	0, 1.5 or 3 (meters)	Cable length setting can be set to any floating-point number between 0 m and 3.0 m, but will be coerced to 0, 1.5, or 3 m.
	KI_CVU_COMPENSATE	[1,1,1] To each item: 0=OFF 1=ON	Set open_comp, short_comp, and load_comp in one command. Values must be list type and have three items.
	KI_CVU_OPEN_COMPENSATE	0 = OFF 1 = ON	Enables or disables compensation constants for open, short, and load.
	KI_CVU_SHORT_COMPENSATE		
	KI_CVU_LOAD_COMPENSATE		
	KI_CVU_FILTER_FACTOR	0 to 100	Sets the custom speed filter factor.
	KI_CVU_MEASURE_SPEED	KI_CVU_SPEED_FAST = 0 KI_CVU_SPEED_NORMAL = 1 KI_CVU_SPEED_QUIET = 2 KI_CVU_SPEED_CUSTOM = 3	Sets CVU speed.
	KI_CVU_CUST_SPEED	<value *n*, *n*, *n*> (*n* represents delay_factor, filter_factor, aperture values) delay_factor: 0 to ~100 aperture: 0006 to ~10.002	Selects the custom speed mode and sets parameter values for delay_factor, filter_factor, and aperture. Values must be list type and have three items, for example: [1,1,1].
	KI_CVU_MEASURE_MODEL	KI_CVU_TYPE_ZTH = 0 KI_CVU_TYPE_RJX = 1 KI_CVU_TYPE_CPGP = 2 KI_CVU_TYPE_CSRS = 3 KI_CVU_TYPE_CPD = 4 KI_CVU_TYPE_CSD = 5 KI_CVU_TYPE_RAW = 6	For more information about the CVU mode see measz (on page 3-35).
	KI_CVU_CHANNEL	1 to ~8	Selects CVU card on which subsequent card CVU commands will act.
	KI_CVU_DCV_OFFSET	-30 to ~30	Applies an offset value to the DC low terminal.
	KI_CVU_ACVHI	1 = HCUR/HPOT 2 = LCUR/LPOT	Allows you to define the source terminal (AC only) for the CVU test to be performed. Unless set otherwise, the default AC source terminal is 1.
	KI_CVU_DCVHI	1 = HCUR/HPOT 2 = LCUR/LPOT	Allows you to define the source terminal (DC only) for the CVU test to be performed. Unless set otherwise, the default DC source terminal is HCUR/HPOT.

Model 4200A CVU LPT parameters			Comments
Unit name	**Modifier**	**Value**	
	KI_CVU_MODE	0 or 1	0: Sets the CVU to user mode. 1: Sets the CVU to system mode.

Model 4284 setmode modifiers

Model 4284 LPT parameters			Comments
Unit name	**Modifier**	**Value**	
CMTR1	KI_CVU_CABLE_CORRECT	0, 1.5, or 3	Cable length setting can be set to any floating-point number between 0 m and 3.0 m, but will be coerced to 0, 1.5, or 3 m.
	KI_CVU_OPEN_COMPENSATE KI_CVU_SHORT_COMPENSATE KI_CVU_LOAD_COMPENSATE	0 = OFF 1 = ON	Enables or disables compensation constants for open, short, and load.
	KI_CVU_FILTER_FACTOR	0 to 100	Sets the custom speed filter factor.
	KI_CVU_MEASURE_SPEED	KI_CVU_SPEED_FAST = 0 KI_CVU_SPEED_NORMAL = 1 KI_CVU_SPEED_QUIET = 2 KI_CVU_SPEED_CUSTOM = 3	Sets CVU speed.
	KI_CVU_MEASURE_MODEL	KI_CVU_TYPE_ZTH = 0 KI_CVU_TYPE_RJX = 1 KI_CVU_TYPE_CPGP = 2 KI_CVU_TYPE_CSRS = 3 KI_CVU_TYPE_CPD = 4 KI_CVU_TYPE_CSD = 5 KI_CVU_TYPE_RAW = 6	For more information about the CVU mode see measz (on page 3-35).
	KI_CVU_MODE	0 or 1	0: Sets CVU to user mode. 1: Sets CVU to system mode.
	KI_AGCV_CORRECT_METHOD		Selects the correction mode (single or multi). Scanner I/F should be installed for multi-mode.
		KI_AGCV_CORRECT_METHOD_MULT = 0	Sets the correction mode to MULTI.
		KI_AGCV_CORRECT_METHOD_SING = 1	Sets the correction mode to SINGLE.
	KI_AGCV_TRIG_SOURCE		Selects the trigger mode.
		KI_AGCV_TRIG_INTERNAL = 0	Sets trigger source to internal.
		KI_AGCV_TRIG_HOLD = 1	Sets trigger source to manual.
		KI_AGCV_TRIG_EXTERNAL = 2	Sets trigger source to external connector on the rear panel.
		KI_AGCV_TRIG_BUS = 3	Sets trigger source to GPIB/LAN/USB.
	KI_AGCV_INIT_CONTINUE	0 = OFF (default value; disables automatic trigger state change) 1 = ON (enables automatic trigger state change)	Enables the automatic trigger to change state from the idle state to the wait for trigger state.
	KI_AGCV_DISPLAY_PAGE		Selects the page to be displayed.
		KI_AGCV_DISPLAY_MEAS = 0	Sets displayed page to <MEAS DISPLAY>.
		KI_AGCV_DISPLAY_BNUMBER = 1	Sets displayed page to <BIN No. DISPLAY>.
		KI_AGCV_DISPLAY_BCOUNT = 2	Sets displayed page to <BIN COUNT DISPLAY>.

Model 4284 LPT parameters			Comments
Unit name	**Modifier**	**Value**	
		KI_AGCV_DISPLAY_LIST = 3	Sets displayed page to <LIST SWEEP DISPLAY>.
		KI_AGCV_DISPLAY_MSETUP = 4	Sets displayed page to <MEAS SETUP>.
		KI_AGCV_DISPLAY_CSETUP = 5	Sets displayed page to <CORRECTION>.
		KI_AGCV_DISPLAY_LTABLE = 6	Sets displayed page to <LIMIT TABLE SETUP>.
		KI_AGCV_DISPLAY_LSETUP = 7	Sets displayed page to <LIST SWEEP SETUP>.
		KI_AGCV_DISPLAY_CATALOG = 8	Sets displayed page to <CATALOG>.
		KI_AGCV_DISPLAY_SYSTEM = 9	Sets displayed page to <SYSTEM INFO>.
		KI_AGCV_DISPLAY_SELF = 10	Sets display page to <SELF TEST>.
		KI_AGCV_DISPLAY_MLARGE = 11	Sets page to display measurement results in large characters.
		KI_AGCV_DISPLAY_SCONFIG = 12	Sets displayed page to <SYSTEM CONFIG>.
		KI_AGCV_DISPLAY_SERVICE = 13	Sets displayed page to <SERVICE>.

sintgi/sintgv

This command performs voltage or current measurement sweeps averaged over a user-defined period (usually, one AC line cycle).

Usage

```
sintgi(instName)
sintgv(instName)
```

instName	Instrument name for measurement

Details

Use the `sintgi` or `sintgv` command to set the measurement method for the integrated measurement of current/voltage sweep.

Note that the measurement setup function `sintgi`/`sintgv` must be called before the sweep functions `sweepi`/`sweepv` or `asweepi`/`asweepv` or `bsweepi`/`bsweepv`.

Example

```
#sweep measure current on SMU1
sintgi(SMU1)
#sweep source voltage from 0~5V by 5 steps and with 0.001s delay
swpData = sweepv(SMU1, 0, 5, 5,0.001)

devint()
#post measured current data
ACSPostArrayDouble("Imeas", swpData["savgi_SMU1"], 5)
```

Also see

asweepi/asweepv (on page 3-9)
bsweepi/bsweepv (on page 3-11)
sweepi/sweepv (on page 3-54)

smeasi/smeasv

This command sets the measurement of current or voltage during a sweep..

Usage

```
smeasi(instName)
smeasv(instName)
```

instName	Instrument name for the measurement

Details

Use the `smeasi` or `smeasv` command to set the measurement for the current or sweep. These commands must be called before the sweep functions `sweepi`, `sweepv`, `asweepi`, `asweepv`, `bsweepi`, or `bsweepv`.

Example

```
#sweep measure current on SMU1
smeasi(SMU1)
#sweep source voltage from 0 to 5 V by 5 steps with a 0.001 s delay.
swpData = sweepv(SMU1, 0, 5, 5,0.001)

devint()
#post measured current data
ACSPostArrayDouble("Imeas", swpData["savgi_SMU1"], 5)
```

Also see

asweepi/asweepv (on page 3-9)
bsweepi/bsweepv (on page 3-11)
sweepi/sweepv (on page 3-54)

sweepi/sweepv

This command performs current and voltage sweeps and returns the measured data based on the measurement setup.

Usage

```
sweepi(instName, startVal, endVal, stepNo, stepDelay)
sweepv(instName, startVal, endVal, stepNo, stepDelay)
```

instName	Instrument name for forcing
startVal	Start value of sweep force
endVal	End value of sweep force
stepNo	Step number of sweep
stepDelay	Delay before every sweep step
return	{"savgi_SMUX":[data_1, data_2,…, data_StepNo]} {"savgv_SMUX":[data_1, data_2,…, data_StepNo]} {"smeasi_SMUX":[data_1, data_2,…, data_StepNo]} {"smeasv_SMUX":[data_1, data_2,…, data_StepNo]} {"singti_SMUX":[data_1, data_2,…, data_StepNo]} {"singtv_SMUX":[data_1, data_2,…, data_StepNo]} {"rtfary":[value_1, value_2,…, value_StepNo]}

Details

Use the `sweepi` or `sweepv` command to perform current or voltage sweep forcing with a defined start value, end value, step number, and step delay. This returns the measured data according to the measurement setup `savgi`, `savgv`, `sintgi`, `sintgv`, `smeasi`, or `smeasv`. If `rtfary` is called, the return value includes the programmed force values of the sweep.

> **NOTE**
>
> The measurement setup function for `savgi`, `savgv`, `sintgi`, `sintgv`, `smeasi`, or `smeasv` must be called before `sweepi` or `sweepv`. The force values return function, `rtfary`, must be called before `asweepi` or `asweepv`.

Example

```
#sweep measure current on SMU1
smeasi(SMU1)
#return force value of sweep
rtfary()
#sweep source voltage from 0~5V by 5 steps and with 0.001s delay
swpData = sweepv(SMU1, 0, 5, 5,0.001)

devint()
#post measured current data
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#post forced voltage value
ACSPostArrayDouble("Vsource", swpData["rtfary], 5)
```

Also see

savgi/savgv (on page 2-22)
sintgi/sintgv (on page 3-52)
smeasi/smeasv (on page 3-53)
rtfary (on page 3-38)

srangei/srangev

This command selects the current or voltage source range and prevents the selected instrument from autoranging.

Usage

`srangei(`*`SMUX`*`, `*`value`*`)`
`srangev(`*`SMUX`*`, `*`value`*`)`

SMUX	The instrument identification code of the instrument; for example, `SMU1`
value	The current or voltage source range

Details

Using the `srangei` or `srangev` command to select the range eliminates the time required for autoranging.

Example

```
#set the current source range on SMU1 to 1e-3A
srangei(SMU1, 1e-3)
#set the voltage source range on SMU2 to 20V
srangev(SMU2, 20)
```

Also see

None

trigil/trigvl/trigig/trigvg

This command sets the level for a break used in the sweep functions `bsweepi` and `bsweepv`.

Usage

```
trigil(instName, level)
trigvl(instName, level)
trigig(instName, level)
trigvg(instName, level)
```

instName	Instrument name for forcing
level	Level of the tested data to break the sweep

Details

When the level is exceeded by the tested data, the sweep breaks as follows:

- `trigil`: When the tested current is lower than the level.
- `trigvl`: When the tested voltage is lower than the level.
- `trigig`: When the tested current is greater than the level.
- `trigvg`: When the tested voltage is greater than the level.

These trigger functions must be called before `bsweepi`/`bsweepv`.

Example

```
#List to store force break value.
BVresult = []
#Set trigger of current low level.
trigil(SMU1, 1e-11)
#Measure current on SMU1.
smeasi(SMU1)
#Return force value of sweep.
rtfary()
#Sweep source voltage from 0~5V by 5 steps, with 0.001s delay, breaks according to the
   trigger level setting.
swpData = bsweepv(SMU1, 0, 5, 5, 0.01, BVresult)
devint()
#Post force break value.
ACSPostArrayDouble("BV", BVresult[0])
#Post measured current data.
ACSPostArrayDouble("Imeas", swpData["smeasi_SMU1"], 5)
#Post forced voltage value.
ACSPostArrayDouble("Vsource", swpData["rtfary"], 5)
```

Also see

bsweepi/bsweepv (on page 3-11)

tstsel

This command is only for the Model 4200A-SCS instrument.

This command enables or disables the test station.

Usage

`tstsel(iStatus)`

iStatus	State of the test station; `1` = enabled, `0` = disabled

Details

The `tstsel(1)` command is normally called at the beginning of a test program to gain control of the test station before running any test control functions..

The `tstsel(0)` can be called at any position of a test program. To relinquish control of the test station, the `tstsel(0)` command is called.

To regain control of the test station, the `tstsel(1)` command is called. This must be done before any subsequent test control functions are performed.

Example

```
#enable control of the test station
tstsel(1)
forcev(SMU1, 1)
Id = measi(SMU1)
#disable control of the test station
tstsel(0)
```

Also see

None

ACS post data and log commands

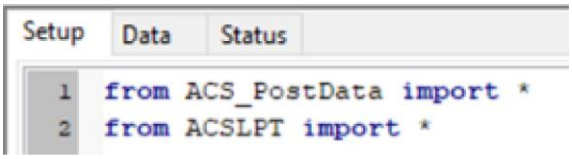
NOTE

Before using the ACS post data and log commands, you need to import `ACS_PostData` to the header of a Python test module (PTM). For more information about importing a header, refer to Creating PTM or STM test libraries and modules (on page 1-2).

NOTE

When you create a new PTM in the test tree, headers are created (`ACS_PostData` and `ACS LPT`), as seen in the following figure. Additionally, you have the option to import a header after the PTM is created.

Figure 30: Import ACSLPT



NOTE

The ACS LPT commands and post data and log commands are listed in alphabetical order.

ACSPostArrayDouble

This command posts an array to the data panel as a double data type.

Usage

```
ACSPostArrayDouble(name, array, size)
```

name	The name of the output data string displayed in the data panel
array	The array posted in double data type
size	The size of array posts as a double data type

Example

```
Imeas = [1.31e-6, 1.32e-6, 1.33e-6]
ACSPostArrayDouble("Imeas",Imeas, 3)
```

Also see

None

ACSPostArrayFloat

This command posts an array to the data panel in float data type.

Usage

```
ACSPostArrayFloat(name, array, size)
```

name	The name of the output data string displayed in the data panel
array	The array posted in float data type
size	The size of the array posted in float data type

Details

Example

```
Vmeas = [2.1, 2.2, 2.3]
ACSPostArrayFloat("Vmeas", Vmeas, 3)
```

Also see

None

ACSPostArrayInt

This command posts an array to the data panel as an integer data type.

Usage

```
ACSPostArrayInt(name, array, size)
```

name	The name of the output data string displayed in the data panel
array	The array posted in an integer data type
size	The size of the array posted in float data type

Details

Example

```
Num_meas = [1, 2, 3, 4, 5]
ACSPostArraryInt("Num_meas", Num_meas, 5)
```

Also see

None

ACSPostDataDouble

This command is used to post data to the data panel as a double data type.

Usage

```
ACSPostDataDouble(name, data)
```

name	The name of the output data string displayed in the data panel
data	The data posted as a double data type

Example

```
ACSPostDataDouble("Imeas", 1.34e-6)
```

Also see

None

ACSPostDataFloat

This command is used to post an array to the data panel as a float data type.

Usage

`ACSPostDataFloat(`*name*`, `*data*`)`

name	The name of the output data string displayed in the data panel
data	The array posted in float data type

Example

```
ACSPostDataFloat("Vmeas", 2.2)
```

Also see

None

ACSPostDataInt

This command is used to post an array to the data panel as an integer data type.

Usage

`ACSPostDataInt(`*name*`, `*data*`)`

name	The name of the output data string displayed in the data panel
data	The data posted in an integer data type

Example

```
ACSPostDataInt("Num_meas", 3)
```

Also see

None

ACSPostGdata

This command is used to define global data that is accessed in the UAP PTM.

Usage

```
ACSPostGdata(name, value)
```

name	The name of the global output data string that is displayed in the UAP PTM
value	The value of the global data

Example

```
ACSPostGdata("global_Vsource", 3.5)
```

Also see

None

logCritical

This command posts critical messages in the log window.

Usage

```
logCritical(msg, color=useColor, title='messageTitle')
```

msg	The message string displayed in the log window
useColor	Enable or disable critical message color: ■ Enable color: `true` ■ Disable color: `false`
messageTitle	The title of the critical message

Example

```
msg = "The SMU%d is not found in the system." %smuNum
logCritical(msg, color=True, title='CRITICAL')
```

Displays a critical message using color.

Also see

None

logError

This command is used to post the error message in the log window.

Usage

`logError(`*`msg`*`, color=`*`useColor`*`)`

msg	The message string displayed in the log window
useColor	Enable or disable error code color: ■ Enable color: `true` ■ Disable color: `false`

Example

```
msg = "The value for Vforce %g is out of limits."%Vforce [1]
logError(msg, color=True)
```

Display the message with color applied.

Also see

None

logInfo

This command posts an information message in the log window.

Usage

`logInfo(`*`msg`*`)`

msg	The message string to be displayed in the log window

Example

```
msg = "The diagnostic of SMU%d is passed"%smuNum
logInfo(msg)
```

Also see

None

PTM examples

ACS LPT using example: vgsid1

Function: `vgsid1(DrainSMU, DrainPin, GateSMU, GatePin, SourceSMU, SourcePin, BulkSMU, BulkPin, GateVStart, GateVStop, numberofpoint, SweepDelay, DrainV, SourceV, BulkV, RangeDrainI, ComplianceDrainI, StoponCompliance, NPLC)`

```
##outputlist = GateV,DrainI,Time##
"""
USRLIB INFORMATION
        VISIBILITY : NOT_HIDDEN
        GUI : True
        GroupNO : 1
        AUTHOR :
        VERSION :
        MODULES : ['vgsid1']
END USRLIB INFORMATION
USRLIB HELP

END USRLIB HELP
"""
from Keithley import *
from ACS_PostData import *
from ACSLPT import *
from ptmlpt.constantlpt import *
from math import *
Get4200HWCtrl()

def vgsid1(DrainSMU='SMU1', DrainPin=1, GateSMU='SMU2', GatePin=2,
   SourceSMU='GNDU', SourcePin=3, BulkSMU='GNDU', BulkPin=4, GateVStart=0,
   GateVStop=0.3, numberofpoint=21, SweepDelay=0.001, DrainV=0.1, SourceV=0,
   BulkV=0, RangeDrainI=1, ComplianceDrainI=0.1, StoponCompliance=0, NPLC=1):
    """
    VISIBILITY: NOT_HIDDEN
    AUTHOR : Keithley Instruments
    VERSION : 1.0
    TYPE : standard
    INPUTLIST
        #Name, Type, Default Value, Min Value, Max Value, Unit
        DrainSMU,enum,'SMU1',('SMU1','SMU2','SMU3','SMU4'),,
        DrainPin,int,1,1,,
        GateSMU,enum,'SMU2',('SMU1','SMU2','SMU3','SMU4'),,
        GatePin,int,2,1,,
        SourceSMU,enum,'GNDU',('SMU1','SMU2','SMU3','SMU4','GNDU'),,
        SourcePin,int,3,1,,
        BulkSMU,enum,'GNDU',('SMU1','SMU2','SMU3','SMU4','GNDU'),,
        BulkPin,int,4,1,,
        GateVStart,double,0,0.0,,V
        GateVStop,double,0.3,0,,V
        numberofpoint,int,21,1,,
```

```
        SweepDelay,double,0.001,0,,
        DrainV,double,0.1,,,V
        SourceV,double,0,,,V
        BulkV,double,0,,,V
        RangeDrainI,double,1,-10,10,A
        ComplianceDrainI,double,0.1,-0.1,0.1,A
        StoponCompliance,int,0,0,1,
        NPLC,double,1,0.001,25,
    END INPUTLIST
    OUTPUTLIST
        #Name, Type, DefaultValue, Unit
    END OUTPUTLIST
    DESCRIPTION
    ---DrainSMU:
    ---DrainPin:
    ---GateSMU:
    ---GatePin:
    ---SourceSMU:
    ---SourcePin:
    ---BulkSMU:
    ---BulkPin:
    ---GateVStart:
    ---GateVStop:
    ---numberofpoint:
    ---SweepDelay:
    ---DrainV:
    ---SourceV:
    ---BulkV:
    ---RangeDrainI:
    ---ComplianceDrainI:
    ---StoponCompliance:
    ---NPLC:
    END DESCRIPTION
    """

    GateV=[]
    DrainI=[]
    Time_meas=[]
    tstsel(1)

    #Some input checking is needed
    if  GateVStart < -200 or GateVStart > 200:
        return INVAL_PARAM
    if  GateVStop < -200 or GateVStop > 200:
        return INVAL_PARAM
    if  numberofpoint < 1 or numberofpoint > 4096:
        return INVAL_PARAM
    if  SweepDelay < 0 or SweepDelay > 100:
        return INVAL_PARAM
    if  DrainV < -200 or DrainV > 200:
        return INVAL_PARAM
    if  SourceV < -200 or SourceV > 200:
        return INVAL_PARAM
    if  BulkV < -200 or BulkV > 200:
        return INVAL_PARAM
    if RangeDrainI < 1 or RangeDrainI > 12:
        return INVAL_PARAM
```

```
    if ComplianceDrainI < -0.1 or ComplianceDrainI > 0.1:
        return INVAL_PARAM

    #Switch Matrix connection
    '''
    clrcon()
    if  GatePin > 0:
        conpin(GateSMU,GatePin)
    if DrainPin > 0:
        conpin(DrainSMU,DrainPin)
    if  SourcePin > 0:
        conpin(SourceSMU,SourcePin)
    if  BulkPin > 0:
        conpin(BulkSMU,BulkPin)
    '''

    #Set the SMUs range
    rangei(GateSMU,0.1)
    rangei(BulkSMU,0.1)
    rangei(SourceSMU,0.1)
    setauto(DrainSMU)
    limiti(DrainSMU, ComplianceDrainI)

    #best fix for voltage range
    if fabs(SourceV) < 0.2:
        rangev(SourceSMU, 0.2)
    elif fabs(SourceV) < 2:
        rangev(SourceSMU, 2)
    elif fabs(SourceV) < 20:
        rangev(SourceSMU, 20)
    else:
        rangev(SourceSMU, 200)

    if fabs(BulkV) < 0.2:
        rangev(BulkSMU, 0.2)
    elif fabs(BulkV) < 2:
        rangev(BulkSMU, 2)
    elif fabs(BulkV) < 20:
        rangev(BulkSMU, 20)
    else:
        rangev(BulkSMU, 200)

    if fabs(DrainV) < 0.2:
        rangev(DrainSMU, 0.2)
    elif fabs(DrainV) < 2:
        rangev(DrainSMU, 2)
    elif fabs(DrainV) < 20:
        rangev(DrainSMU, 20)
    else:
        rangev(DrainSMU, 200)

    if fabs(GateVStart) > fabs(GateVStop):
        temp = fabs(GateVStart)
    else:
        temp = fabs(GateVStop)
    if temp < 0.2:
        rangev(GateSMU, 0.2)
```

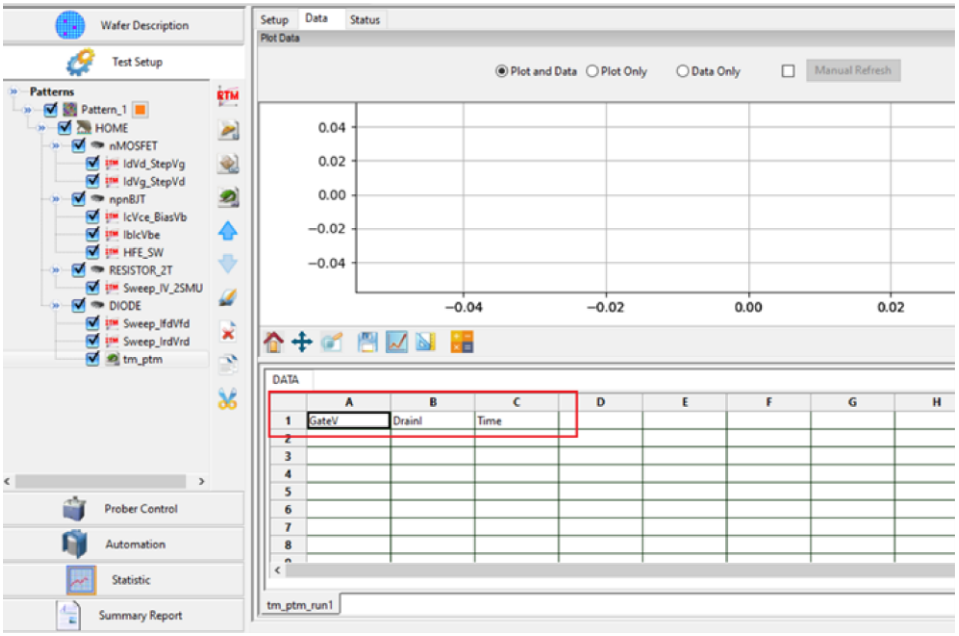
```
    elif temp < 2:
        rangev(GateSMU, 2)
    elif temp < 20:
        rangev(GateSMU, 20)
    else:
        rangev(GateSMU, 200)

    #autorange
    if RangeDrainI == 1:
        setauto(DrainSMU)
    #limited auto 10pA
    elif RangeDrainI == 2:
        lorangei(DrainSMU, 1e-11)
    #limited auto 100pA
    elif RangeDrainI == 3:
        lorangei(DrainSMU, 1e-10)
    #limited auto 1nA
    elif RangeDrainI == 4:
        lorangei(DrainSMU, 1e-9)
    #limited auto 10nA
    elif RangeDrainI == 5:
        lorangei(DrainSMU, 1e-8)
    #limited auto 100nA
    elif RangeDrainI == 6:
        lorangei(DrainSMU, 1e-7)
    #limited auto 1uA
    elif RangeDrainI == 7:
        lorangei(DrainSMU, 1e-6)
    #limited auto 10uA
    elif RangeDrainI == 8:
        lorangei(DrainSMU, 1e-5)
    #limited auto 100uA
    elif RangeDrainI == 9:
        lorangei(DrainSMU, 1e-4)
    #limited auto 1mA
    elif RangeDrainI == 10:
        lorangei(DrainSMU, 1e-3)
    #limited  auto 10mA
    elif RangeDrainI == 11:
        lorangei(DrainSMU, 1e-2)
    #limited auto 100mA
    elif RangeDrainI == 12:
        lorangei(DrainSMU, 0.1)
    #limited auto 10mA
    else:
        lorangei(DrainSMU, 1e-2)

    #set integration time
    setmode(GateSMU, KI_INTGPLC, NPLC)

    #Activate the range
    if  SourceSMU!=GNDU:
        forcev(SourceSMU, SourceV)
    if  BulkSMU!=GNDU:
        forcev(BulkSMU,BulkV)
    forcev(GateSMU,GateVStart)
    forcev(DrainSMU,DrainV)
```

```
    idummy = measi(DrainSMU)
    enable(TIMER1)

    #sweep setup
    if  numberofpoint>1:
        for index1 in range(numberofpoint):
            GateV_tmp =  GateVStart+(GateVStop-GateVStart)*index1/
                (numberofpoint-1)
            GateV.append(GateV_tmp)
            forcev(GateSMU,GateV_tmp)
            delay(int(SweepDelay*1000))
            DrainI_tmp = intgi(DrainSMU)
            if DrainI_tmp > ComplianceDrainI:
                break
            DrainI.append(DrainI_tmp)
            Time_meas.append(imeast(TIMER1))
    else:
        forcev(GateSMU, GateVStart)
        GateV.append(GateVStart)
        delay(int(SweepDelay*1000))
        DrainI.append(intgi(DrainSMU))
        Time_meas.append(imeast(TIMER1))

    #check compliance
    Dstatus = getstatus(DrainSMU, KI_COMPLNC)
    if  Dstatus == 2:
        return  KI_RANGE_COMPLIANCE
    if  Dstatus == 4:
        return  KI_COMPLIANCE
    devint( )

    #clrcon(MTRX1)
    #test finished
    for index2 in range(numberofpoint):
        ACSPostDataDouble("GateV", GateV[index2])
        ACSPostDataDouble("DrainI", DrainI[index2])
        ACSPostDataDouble("Time", Time_meas[index2])

    return GateV,DrainI,Time_meas
```

NOTE

Make sure that the Python test module has an output parameter list in the data panel. Otherwise, it must be specified, as shown in the above example (`##outputlist = output_name1, output_name2, ……##`). Also, the following graphic identifies where you can output the specified data.

Figure 31: PTM output list



Section 4

ACS TSP user library

In this section:

Series 2600 library

The ACS software has a test library of commands (`26Library`) for the Series 2600B SourceMeter® instruments. This library is in the following directory: `\\ACS\library\26Library`. It includes device test libraries and a wafer-level reliability (WLR) library.

The tables in this section summarize all the test modules in the device test and wafer-level reliability (WLR) libraries.

Bipolar junction transistor (BJT) test library		
Three_term_BJT_BVCBO	Three_term_BJT_BVCEI	Three_term_BJT_BVCEO
Three_term_BJT_BVCES	Three_term_BJT_BVCEV	Three_term_BJT_BVEBO
Three_term_BJT_BVECO	Three_term_BJT_HFE_sw	Three_term_BJT_HFE_tral
Three_term_BJT_IBCO	Three_term_BJT_IBEO	Three_term_BJT_ibicvbe
Three_term_BJT_ibvbe	Three_term_BJT_ICBO	Three_term_BJT_ICEO
Three_term_BJT_ICES	Three_term_BJT_ICEV	Three_term_BJT_icvcb
Three_term_BJT_icvce_biasIB	Three_term_BJT_icvce_biasVB	Three_term_BJT_icvce_stepib
Three_term_BJT_icvce_stepvb	Three_term_BJT_IEBO	Three_term_BJT_IECO
Three_term_BJT_ieveb	Three_term_BJT_VBCO	Three_term_BJT_VCE

MOSFET Test Library		
Four_term_MOSFET_BVDSS	Four_term_MOSFET_BVDSV	Four_term_MOSFET_BVGDO
Four_term_MOSFET_BVGDS	Four_term_MOSFET_BVGSO	Four_term_MOSFET_IDL
Four_term_MOSFET_IDS_ISD	Four_term_MOSFET_idvd	Four_term_MOSFET_idvd_vg
Four_term_MOSFET_idvg	Four_term_MOSFET_idvg_vd	Four_term_MOSFET_idvg_vsub
Four_term_MOSFET_IGL	Four_term_MOSFET_igvg	Four_term_MOSFET_ISL
Four_term_MOSFET_isubvg	Four_term_MOSFET_Vth_ci	Four_term_MOSFET_Vth_ex
Four_term_MOSFET_Vth_llsq	Four_term_MOSFET_Vth_sense	

Diode Test Library		
Diode_DynamicZ	Diode_DynamicZ_I1I2	Diode_Ifd_Vfd
Diode_Ifd_Vfd_vsweep	Diode_Ileakage_Vrd	Diode_Ird_Vrd_vsweep
Diode_Vrd_Ird	Diode_Vbr_Ird	Diode_Vfd_Ifd

Resistor Test Library	
Resistor_single	Resistor_sweep
VDP_Resistivity	

Create a library without Script Editor

You can use test script language on the Keithley Instruments Series 2600B System SourceMeter® instrument or the linear parametric test library (LPT library) to create a new library.

NOTE

You must use the syntax of the script that follows the rules for the Lua programming language (see the following information).

The script must use the `postdata`, `postbuffer`, or `posttable` function to retrieve data from a Series 2600B instrument. Refer to LACS TSB LPT library reference (on page 2-1) for more information. For examples, refer to the following directory `\\ACS\Library\26Library`.

To design a test library with a graphical user interface:

1. The first line must be the name of the `.xrc` (user interface) file. The `.xrc` file must be placed in the `\\ACS\library\26Library\xrc` folder. ACS then loads the user interface file automatically when importing the script file.

   `----<<xrc=HCI.xrc>>----`

2. The types of input variables must be:
   - instid (SMU input)
   - string
   - double
   - integer
   - table
3. You can set a default value for every input variable. You can also set the input range for double and integer-type input variables.

`instid smu_S=SMU3`	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`, `KI_GND`
`double vg_stress=-2.0 in [-40,40]`	Gate stress voltage; –40 = vg_stress = 40
`double V_rd=0 in ['',0]`	Reverse voltage; V_rd ≤ 0
`double meas_delay=0 in [0,]`	Measure delay after stress is off; meas_delay ≥ 0
`integer navg=1 in [1,20]`	Points for average; average = 1, 2, 3,…19, 20
`table t_array={1,2,5,10,20,50,100}`	Stress time array

4. The input variables must be defined in the first section of the test script, after the .xrc line, listed between `--INPUT--` and `--END of INPUT--`.

INPUT	
instid CSMU=SMU3	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`
double Vb_stop=1.2	Stop voltage (units: V)
double Vb_points=100	Sweep points
integer resetflag=1 in [0,1]	`1`: Resets instruments after test `0`: Does not reset instruments
END OF INPUT	

5. The Call function must start with a `--CALL--` line, then assign a value for every input variable and call test function.

NOTE

Refer to the following directory for examples: `\\ACS\Library\26Library\WLR`.

Create a library using Script Editor

You can also use the Script Editor in ACS to create a new library. For more information about how to use the script editor, refer to Creating PTM or STM test libraries and modules (on page 1-2).

When you use the script editor, the library is saved automatically to this directory on your computer: `\\ACS\library\26Library\TSPLib`.

The .xrc user interface is saved automatically to this directory on your computer: `\\ACS\library\26Library\TSPLib\xrc`.

Device library

The following topics describe the device libraries:

BJT library overview

The BJT test library is used to test parameters of a bipolar junction transistor (BJT) such as breakdown voltage, amplify times, reverse current, and Gummel plot. You can use the 2600B library (in the `\\ACS\Library\26Library` folder) with a Series 2600B instrument to create test script files based on the Series 2600B Linear Parametric Test (LPT) Library. You also can use the 4200A-SCS library (in the `\\ACS\Library\42Library` folder) with a Model 4200A-SCS to create Keithley User Library Tool (KULT) files based on the Model 4200A-SCS LPT Library.

The bipolar junction transistor (BJT) library components are in the following directories:

- `\\ACS\Library\26Library\Parametric\BJT`
- `\\ACS\Library\42Library\Parametric\S4200ParLib\src`

Three_term_BJT_BVCBO

This module gets the collector-emitter breakdown voltage of a three-terminal BJT with the emitter open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the emitter and apply the specified current to the collector. The base connects to ground.

Intended results

Get the collector-emitter breakdown voltage.

Three_term_BJT_BVCEI

This module tests the collector-emitter breakdown voltage of a three-terminal BJT with a biased current forced at the base.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply the specified current to the collector and set the base bias current. The emitter connects to ground.

Intended results

Get the collector-emitter breakdown voltage.

Three_term_BJT_BVCEO

This module gets the collector-emitter breakdown voltage of a three-terminal BJT with the base open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the base and apply the specified current to the collector. The emitter connects to ground.

Intended results

Get the collector-emitter breakdown voltage.

Three_term_BJT_BVCES

This module tests the collector-emitter breakdown voltage of the three-terminal BJT with a short at the base-emitter.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply the specified current to the collector. The base and emitter connect to ground.

Intended results

Get the collector-emitter breakdown voltage.

Three_term_BJT_BVCEV

This module tests the collector-emitter breakdown voltage of a three-terminal BJT with a biased voltage forced at the base.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply the current to the collector and bias voltage at the base. The emitter is connected to ground.

Intended results

Get the collector-emitter breakdown voltage.

Three_term_BJT_BVEBO

This module tests the emitter-base breakdown voltage of a three-terminal BJT with the collector open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the collector and set the emitter to the specified current. Connect the base to ground.

Intended results

Get the emitter-base breakdown voltage.

Three_term_BJT_BVECO

This module tests the emitter-collector breakdown voltage of a three-terminal BJT with the base open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the base and apply the specified current to the emitter. Connect the collector to ground.

Intended results

Get the emitter-collector breakdown voltage.

Three_term_BJT_HFE_sw

This module tests the DC current gain of a three-terminal BJT with a collector voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection, apply a sweep voltage on the collector and apply a bias voltage to the base. The emitter is connected to ground (if it is not connected to ground, be sure to apply a bias voltage on the emitter).

Intended results

Get the collector current, base current, and DC current gain based on the collector sweep voltage.

Three_term_BJT_HFE_tral

This module tests the DC current gain of a three-terminal BJT with a traditional test method.

Instrument

Keithley Instruments Series 2600B

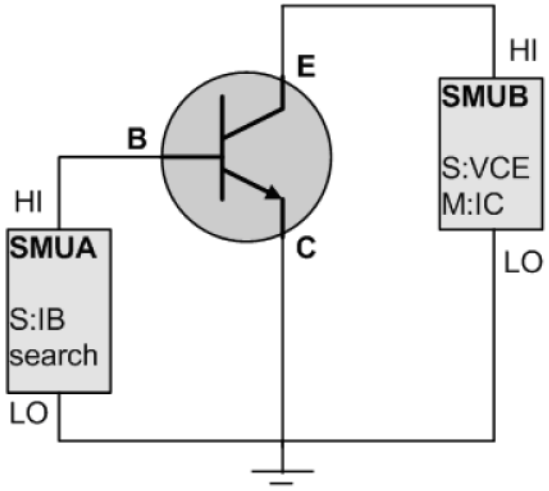
Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection, apply a voltage on the collector and apply a current to the base. The emitter connects to ground. If the emitter is not connected to ground, there is a specified voltage applied.

Figure 32: Three_term_BJT_HFE_tral pin connection



Use this technique:

1. Set base current level, measure collector current, and verify that it equals the expected target.
2. If the collector current is not at the expected target, repeat step 1.
3. Find the target collector current and record the base current.
4. Calculate HFE=IC_tar/IB.

Intended results

Get the DC current gain.

Three_term_BJT_IBCO

This module tests the base-collector current of a three-terminal BJT with the emitter open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the emitter, apply a voltage on the base, and apply voltage to the collector (if not connected to ground).

Intended results

Get the base-collector current.

Three_term_BJT_IBEO

This module tests the base-emitter current of a three-terminal BJT with the collector open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the base connection, apply a sweep voltage on the collector, and apply a bias voltage on the emitter. The base is usually connected to ground, but can be set to a bias voltage.

Intended results

Get the base-emitter current.

Three_term_BJT_ibicvbe

This module tests the base current and collector current of a three-terminal BJT with a specified base voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection, apply a sweep voltage on the base and apply a bias voltage on the collector. The emitter is usually connected to ground, but can be set to a specific bias voltage.

Use this technique:

1. Measure the base current and collector current of the BJT.
2. Get the Ib-Vbe and Ic-Vbe curve.
3. Get the Gummel plot if the axis properties of data plot have changed (logarithm instead of right-angle coordinate).

Intended results

Get the base current and collector current.

Three_term_BJT_ibvbe

This module tests the base current of a three-terminal BJT with a specified base voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection, apply a sweep voltage on the base, and apply a bias voltage on the collector. The emitter is usually connected to ground, but can be set to a bias voltage.

Intended results

Intended results: Get the measured base current according to the base voltage sweep results.

Three_term_BJT_ICBO

This module tests the collector-base cut off current of a three-terminal BJT with the emitter open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the emitter and apply a voltage to the collector. The base is connected to ground.

Intended results

Intended results: Get the collector-base to cut off current.

Three_term_BJT_ICEO

This module tests the collector-emitter cut off current of a three-terminal BJT with the base open.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the base, apply a voltage to the collector, and connect the emitter to ground.

Intended results

Get the collector-emitter to cut off current.

Three_term_BJT_ICES

This module tests the collector-emitter cut-off current of a three-terminal BJT with the base-emitter shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply a voltage to the collector and connect the emitter and base to ground.

Intended results

Intended results: Get the collector-emitter to cut off current.

Three_term_BJT_ICEV

This module tests the collector-emitter cut-off current of a three-terminal BJT with a bias voltage on the base.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply a voltage to the collector, apply a bias voltage on the base, and connect the emitter to ground.

Intended results

Intended results: Get the collector-emitter to cut off current.

Three_term_BJT_icvcb

This module tests the collector-current of a three-terminal BJT with a specified collector voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection, apply a sweep voltage on the collector, apply a bias voltage on the base. The emitter is usually connected to ground, but can be set to a bias voltage.

Intended results

Get the measured collector current according to the collector voltage sweep.

Three_term_BJT_icvce_biasIB

This module tests a series of collector-current and collector-emitter voltage (Ic-Vce) curves of a three-terminal BJT while stepping the base current.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection (connect the emitter to ground), step the base current, and sweep the collector voltage.

Intended results

Get the measured collector current according to the base step current and collector sweep voltage.

Three_term_BJT_icvce_biasVB

This module tests a series of Ic-Vce curves of a three-terminal BJT while stepping the base voltage.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection (connect the emitter to ground), step the base voltage, and sweep the collector voltage.

Intended results

Get the measured collector current according to the base step voltage and collector sweep voltage.

Three_term_BJT_icvce_stepib

This module tests a series of Ic-Vce curves of a three-terminal BJT while stepping the base current.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection (connect the emitter to ground), step the base current, and sweep the collector voltage.

Intended results

Get the measured collector current according to the base step current and collector sweep voltage.

Three_term_BJT_icvce_stepvb

This module tests a series of Ic-Vce curves of a three-terminal BJT while stepping the base voltage.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the emitter connection (connect the emitter to ground), step the base voltage, and sweep the collector voltage.

Intended results

Get the measured collector current according to the base step and collector sweep voltage.

Three_term_BJT_IEBO

This module tests the emitter-base cut-off current of a three-terminal BJT with the collector open.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the collector and apply a voltage to the emitter. The base connects to ground.

Intended results

Intended results: Get the emitter-base cut-off current.

Three_term_BJT_IECO

This module tests the emitter-collector current of a three-terminal BJT with the base open.

Instrument

Keithley Instruments Model 4200A-SCS

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the base and apply a voltage to the emitter. The collector is usually connected to ground if voltage is not applied.

Intended results

Get the emitter-collector current.

Three_term_BJT_ieveb

This module tests the emitter-current of a three-terminal BJT with a specified emitter voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Sharing the base connection, apply a sweep voltage on the emitter, and apply a bias voltage on the collector. Connect the base to ground if voltage is not applied.

Intended results

Get the measured emitter-current based on the emitter voltage sweep.

Three_term_BJT_VBCO

This module tests the base-collector voltage of a three-terminal BJT with the emitter open.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Open the emitter and apply a current to the base. The emitter usually connects to ground, but can be set to a bias voltage.

Intended results

Get the base-collector voltage.

Three_term_BJT_VCE

This module tests the collector-emitter voltage of a three-terminal BJT.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Three-terminal BJT

Pin connections

Apply voltage on the base, set the collector-current to a target level, and connect the emitter to ground.

Intended results

Get the collector-emitter voltage.

MOSFET library overview

The MOSFET library components are in the following directories on your computer:

- `\\ACS\Library\26Library\Parametric\MOSFET`
- `\\ ACS\Library\42Library\Parametric\S4200ParLib\src`

You can test parameters of a MOSFET such as `gmlin`, `gmsat`, `idvd`, `idvg`, `igvg`, `Vtci`, and `Vtex` with the MOSFET parameter library. You can use the 2600B library (in the `\\ACS\Library\26Library` folder) with a Series 2600B instrument to create test script files based on the Series 2600B Linear Parametric Test (LPT) Library. You also can use the 4200A-SCS library (in the `\\ACS\Library\42Library` folder) with a Model 4200A-SCS to create Keithley User Library Tool (KULT) files based on the Model 4200A-SCS LPT Library.

Four_term_MOSFET_BVDSS

This module tests the drain-source breakdown voltage of a four-terminal MOSFET with the gate-source shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Apply a breakdown current on the drain and connect the bulk to ground. If the bulk is not connected to ground, force 0 V. The gate and source are connected to ground; if they are not connected to ground, force 0 V.

Intended results

Get the breakdown voltage between the drain and source with gate-source shorted.

Four_term_MOSFET_BVDSV

This module tests the drain-source breakdown voltage of a four-terminal MOSFET with the gate biased.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

The source and bulk are connected to ground, and the gate is biased. Apply a breakdown current on the drain.

Intended results

Get the breakdown voltage between the drain and source with the gate biased.

Four_term_MOSFET_BVGSO

This module tests the gate-source breakdown voltage of a four-terminal MOSFET with the drain opened.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Open the drain and connect the bulk and source to ground. Apply a breakdown current on the gate.

Intended results

Get the breakdown voltage between the gate and source with the drain opened.

Four_term_MOSFET_BVGDS

This module tests the gate-drain breakdown voltage of a four-terminal MOSFET with the source-drain shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Connect the source and drain to ground. Apply a breakdown current on the gate.

Intended results

Get the breakdown voltage between the gate and drain with source-drain shorted.

Four_term_MOSFET_BVGDO

This module tests the gate-drain breakdown voltage of a four-terminal MOSFET with the source opened.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Open the source and connect the bulk and drain to ground. Apply a breakdown current on the gate.

Intended results

Get the breakdown voltage between the gate and drain when the source is open.

Four_term_MOSFET_IDL

This module measures the drain leakage-current with the gate-source shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Short the gate and source. Apply a voltage on the drain, with the bulk, gate, and source connected to ground.

Intended results

Get the drain leakage-current with the gate-source shorted.

Four_term_MOSFET_IDS_ISD

This module measures the drain-source and the source drain-current of a four-terminal MOSFET with the gate biased.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Apply voltage separately on the gate, source, and drain. The bulk is usually connected to ground, but can be set with a bias voltage.

Intended results

Get the measured drain-source and source drain-current with the gate biased.

Four_term_MOSFET_idvd

This module tests the drain-current of a four-terminal MOSFET at a specified drain-voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the gate and sweep the drain. Connect the bulk and source to ground if voltage is not applied.

Intended results

Get the measured drain-current at a specified drain-voltage sweep.

Four_term_MOSFET_idvd_vg

This module tests a series of Id_Vd curves for a four-terminal MOSFET that performs on the Series 2600B.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Sweep the drain and step the gate. Connect the bulk and source to ground if voltage is not applied.

Intended results

Get the measured drain-current at a specified drain-voltage sweep.

Four_term_MOSFET_idvg

This module tests the drain-current at a specified gate-voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and sweep the gate. Connect the bulk and source to ground if voltage is not applied.

Intended results

Get the measured drain-current at the gate-voltage sweep.

Four_term_MOSFET_idvg_vd

This module tests the drain-current of a four-terminal MOSFET at a specified gate-voltage sweep while stepping the drain.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Step the drain and sweep the gate. Connect the bulk and source to ground if voltage is not applied.

Intended results

Get the measured drain-current at the gate-voltage sweep.

Four_term_MOSFET_idvg_vsub

This module tests the drain-current of a four-terminal MOSFET at a specified gate-voltage sweep while stepping the bulk.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Step the bulk, sweep the gate, and bias the drain. Connect the source to ground if voltage is not applied.

Intended results

Get the measured drain-current at the gate-voltage sweep.

Four_term_MOSFET_IGL

This module measures the gate leakage-current of a four-terminal MOSFET while the source-drain is shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Apply a voltage on the gate. Connect the source, drain, and bulk to ground.

Intended results

Get the gate leakage-current when the source and drain are shorted.

Four_term_MOSFET_igvg

This module tests the gate-current of a four-terminal MOSFET at a specified gate-voltage sweep when the drain is biased.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and sweep the gate. Connect the bulk and source to ground or apply a voltage.

Intended results

Get the measured gate-current at the gate-voltage sweep.

Four_term_MOSFET_ISL

This module measures the source leakage-current of a four-terminal MOSFET when gate-drain is shorted.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Apply a voltage on the source. Connect the bulk, gate, and drain to ground.

Intended results

Get the source leakage-current when gate-drain is shorted.

Four_term_MOSFET_isubvg

This module tests the bulk-current of a four-terminal MOSFET at a specified gate-voltage sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and bulk, and sweep the gate. Connect the source to ground if voltage is not applied.

Intended results

Get the measured bulk-current at the gate-voltage sweep.

Four_term_MOSFET_Vth_ci

This module geta the constant current threshold voltage of a four-terminal MOSFET.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and sweep the gate. Input the source and bulk voltage when needed. The source and bulk are usually connected to ground for NMOS, and connected to the normal power supply voltage (VDD) for PMOS.

Technique

The constant current threshold voltage is defined in the following table.

Vth_ci=VGS (@ID=1uA.W/L)	NMOS
Vth_ci=VGS (@ID=-0.025uA.W/L)	PMOS

Where: W and L are the gate-width and gate-length as printed on the wafer.

Set a target drain-current Id_tar (Id_tar=1 uA.W/L, or -0.025 uA.W/L), which means it is near the threshold, then search the gate-voltage to make the drain-current equal to Id_tar.

NOTE

The Four_term_MOSFET_Vth_ci measurement technique must determine Vth_ci to within a 1 mV resolution. If the VGS step size is larger than 1 mV, then a linear interpolation method may be used to achieve the 1 mV resolution.

Typical DC bias voltages for Vth_ci measurements are VDS = VDS_lin, VBS = VBB for linear region measurement, or VDS = VDS_sat, (VBS = VBB for saturation region measurement). Typically, for PMOS, VDS_lin = -0.1 V (@VDD=5 V); for NMOS, VDS_lin=0.1 V(@VDD=5 V).

Intended results

Get the constant-current threshold voltage.

Four_term_MOSFET_Vth_ex

This module gets threshold voltage of a four-terminal MOSFET from the maximum slope measurement.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and sweep the gate. Input source and bulk voltage when needed. The source and bulk are usually connected to ground for NMOS, and connected to the normal power supply voltage (VDD) for PMOS.

Technique

The threshold voltage is extrapolated from the measurement of the maximum slope (Gmmax) of the ID-VGS curve, as described below:

Vth_ex=VGS (@Gmmax)-ID(@Gmmax)/Gmmax

Where VGS (@Gmmax) is the gate-voltage at the point of the maximum slope of the ID-VGS curve and ID(@Gmmax) is the drain-current at the point of the maximum slope of the ID-VGS curve. Also, Gmmax is the maximum slope of the ID-VGS curve.

NOTE

DC bias voltages for Vth_ex measurements are VDS = VDS_lin, VBS = VBB for linear measurement.

VDS=VDS_sat, VBS=VBB for saturation. Typically, for PMOS, VDS_lin = -0.1 V(@VDD=5 V); for NMOS, VDS_lin=0.1 V(@VDD=5 V).

Intended results

Get the measured drain-current at the gate-voltage sweep.

Extract the transconductance(Gm) and get the maximum transconductance (Gmmax).

Get the extracted threshold voltage (Vth_ex).

Get the drain-current versus the gate-voltage curve.

Get the Gm versus the drain-current or the Gm versus the gate-voltage curve.

Four_term_MOSFET_Vth_llsq

This module extracts the threshold voltage from the measurement slope using the least-square approximation.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Four-terminal MOSFET

Pin connections

Bias the drain and sweep the gate. Input source and bulk voltage when needed. The source and bulk are usually connected to ground for NMOS, and connected to the normal power supply voltage (VDD) for PMOS.

Technique

The threshold voltage is extrapolated from the measurement of the maximum slope (Gmmax) of the ID-VGS curve, as described below:

Vth_ex=VGS (@Gmmax)-ID(@Gmmax)/Gmmax.

Where VGS(@Gmmax) is the gate-voltage at the point of the maximum slope of the ID-VGS curve and ID(@Gmmax) is the drain-current at the point of the maximum slope of the ID-VGS curve. Also, Gmmax is the maximum slope of the ID-VGS curve.

NOTE

DC bias voltages for Vth_ex measurements are VDS = VDS_lin, VBS = VBB for linear measurement.

VDS=VDS_sat,VBS = VBB for saturation.

Typically, for PMOS, VDS_lin = -0.1 V(@VDD = 5 V); for NMOS,VDS_lin = 0.1 V(@VDD = 5 V).

Intended results

Get the measured drain-current at the gate-voltage sweep.

Extract the transconductance (Gm) and get the maximum transconductance (Gmmax).

Get the extracted threshold voltage (Vth_ex).

Get the drain-current versus gate-voltage curve.

Get the Gm versus the drain-current or the Gm versus the gate-voltage curve.

Four_term_MOSFET_Vth_sense

This module gets the self-biasing threshold voltage (Vth_sel) for a four-terminal MOSFET.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

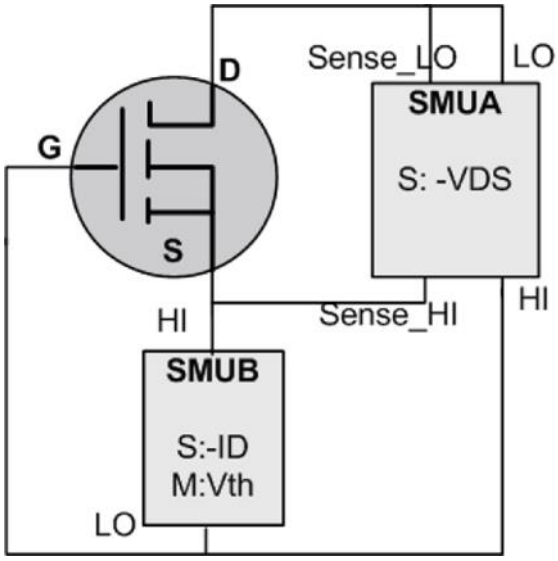
Four- terminal MOSFET

Pin connections

Connect the bulk, source, HI terminal of SMU_B, and sense_HI terminal of SMU_A together. Connect the gate, LO terminal of SMU_B and HI terminal of SMU_A together. Connect the drain, sense_LO, and LO terminal of SMU_A together.

Use two SMUs. For example, SMU_A and SMU_B. SMU_A is in sense mode.

Figure 33: Four-terminal MOSFET Vth_sense pin connections



Technique

Set SMU_B to the drain-current level (IDS).

Set SMU_A to the VDS level.

Measure the VG using SMU_B.

Ensure the threshold voltage equals the gate-voltage.

NOTE

When in sense mode, SMU_A automatically varies the voltage on the gate until VDS is equal to the specified level and at that time measures the gate-voltage. The threshold voltage should equal the measured gate-voltage.

Intended results

Get the self-biasing threshold voltage.

Diode library overview

The Diode library components are in the following directories on your computer:

- `\\ACS\Library\26Library\Parametric\Diode`
- `\\ ACS\Library\42Library\Parametric\S4200ParLib\src`

You can test the parameters of a diode, such as the forward voltage and current, reverse voltage and current, I-V curve, and dynamic impedance with the diode test library. You can use the 2600B library (in the `\\ACS\Library\26Library` folder) with a Series 2600B instrument to create test script files based on the Series 2600B Linear Parametric Test (LPT) Library. You also can use the 4200A-SCS library (in the `\\ACS\Library\42Library` folder) with a Model 4200A-SCS to create Keithley User Library Tool (KULT) files based on the Model 4200A-SCS LPT Library.

Diode_DynamicZ

This module calculates the dynamic impedance based on two forward voltage measurements or two reverse voltage measurements, for example, DynamicZ = (v2 - v1) / (I2 - I1).

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Uses one SMU to force the forward current, while the other terminal is connected to ground.

Intended results

Get dynamic impedance.

Diode_Ifd_Vfd

This module tests the forward current of a diode at a specified forward voltage.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Force a forward voltage on the P terminal. Connect the N terminal to ground.

Intended results

Get the forward current.

Diode_Ifd_Vfd_vsweep

This module tests the forward current with a forward voltage sweep to indicate the forward I-V characteristics of a diode.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Apply a forward sweep voltage to terminal P. Connect the N terminal to ground.

Intended results

Forward the voltage based on the forward current sweep.

Diode_Ileakage_Vrd

This module tests the leakage-current of a diode at a specified reverse voltage.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Apply a forced reverse voltage, or zero voltage to the N terminal. Connect the P terminal to ground.

Intended results

Get the reverse leakage current.

Diode_Ird_Vrd_vsweep

This module tests the reverse current with a reverse voltage sweep to indicate the reverse I-V characteristics of a diode.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Apply a reverse voltage sweep to the N terminal. Connect the P terminal to ground.

Intended results

Reverse the current at each reverse voltage sweep point.

Diode_Vbr_Ird

This module tests the breakdown voltage of a diode at a specified reverse current.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Force a reverse current on the N terminal. Connect the P terminal to ground.

Intended results

Determine the breakdown voltage of a diode.

Diode_Vfd_Ifd

This module tests the forward voltage of a diode.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Use one SMU to force the forward current while the other terminal is grounded. The forward voltage is measured at the current.

Intended results

Determine the forward voltage of a diode.

Diode_Vrd_Ird

This module tests the reverse voltage of a diode at a specified reverse current.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Diode

Pin connections

Force a reverse current to the N terminal. Connect the P terminal to ground.

Intended results

Determine the reverse voltage of a diode.

Resistor library overview

The resistor library components are in the following directories on your computer:

- `\\ACS\Library\26Library\Parametric\Resistor`
- `\\ ACS\Library\42Library\Parametric\S4200ParLib\src`

The resistor test library tests parameters of a resistor (source V, measure I, or source I measure V, 2-wire, or 4-wire). You can use the 2600B library (in the `\\ACS\Library\26Library` folder) with a Series 2600B instrument to create test script files based on the Series 2600B LPT library. You can also use the 4200A library (in the `\\ACS\Library\42Library` folder) with a Model 4200A-SCS to create Keithley User Library Tools (KULT) files, based on the Model 4200A-SCS LPT library.

Resistor_single

This module measures resistance at specified voltage or current stress.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Two-terminal generic device

Pin connections

It supports one or two SMUs, with terminal 1 to do the stress and measure, terminal 2 to be 0 or connected to ground.

Intended results

Resistance reading at voltage or current stress.

Resistor_sweep

This module measures resistance with specified voltage or current sweep.

Instrument

Keithley Instruments Series 2600B

Device under test (DUT)

Two-terminal generic device

Pin connections

Supports one or two SMUs. Terminal 1 is used for stress and measurement; terminal 2 set to 0 or connected to ground.

Intended results

Resistance reading at a specified voltage or current sweep.

WLR library overview

The WLR test library provides certain wafer-level reliability tests on devices with Series 2600B instruments. The HCI, TDDB, and two NBTI tests are available in this library.

There are test script files, based on the Series 2600B LPT library. The wafer-level reliability (WLR) library components are in the `\ACS\Library\26Library\WLR` directory.

The following table summarizes all the test modules in the WLR library. More detailed descriptions of each module follow the tables.

Wafer-level reliability (WLR) Test Library		
HCI	NBTI	NBTI_on_the_fly
TDDB_CCS	TDDB_per_pin	qbd_rmpj
qbd_rmpv	Em_iso_test	NBTI_meas

HCI

This test script runs the HCI test.

Usage

```
HCI(t_mode, t_max, npdec_delta, time_input, SSMU, BSMU, GSMU, DSMU, myNPLC, VSS,
   S_comp, VBB, B_comp, Id_Vg, Vg_start, Vg_stop, Vg_points, G_comp, Vd_lin,
   D_comp, Id_target, Vd_sat, Vd_leak, Vg_leak, Abort_shift, Abort_Vt, Abort_Ig,
   time_interval, Vg_stress, Vd_stress, Vb_stress, G_stress_comp, D_stress_comp)
```

Inputs	
integer t_mode=0 in [0,2]	0: Linear 1: Logarithmic 2: Take input time array
integer t_max=1000 in [0,]	Maximum time for the test; not in use when t_mode is 2
integer npdec_delta=3 in [0,]	The time interval when t_mode is 0; the number of points in one decade when t_mode is 1
table time_input={0,1,2,5,10}	When t_mode is 2 the time array should be input from outside
instid SSMU=KI_GND	SMU1, SMU2, SMU3, ..., SMU64, KI_GND
instid BSMU=KI_GND	SMU1, SMU2, SMU3, ..., SMU64, KI_GND
instid GSMU=SMU1	Gate SMU
instid DSMU=SMU2	Drain SMU
double myNPLC=0.001 in [0.001,25]	Set NPLC value
double VSS=0	Voltage applied on source if not connected to GND
double S_comp=0.1 in [0,]	Source compliance during test and stress (unit: A)
double VBB=0	Voltage applied on substrate if not connected to GND
double B_comp=0.1 in [0,]	Source compliance during test and stress (unit: A)
integer Id_Vg=0 in [0,1]	1: Id_Vg curve is output 0: The curve is not output
double Vg_start=0	If nil, no Vth output; start voltage for sweep on gate (unit: V)
double Vg_stop=1.5	Stop voltage for sweep on gate (unit: V)
integer Vg_points=101 in [0,]	Number of points in the sweep
double G_comp=0.1 in [0,]	Gate compliance during test (unit: A)
double Vd_lin=0.1	Drain voltage in linear district (unit: V)
double D_comp=0.1 in [0,]	Drain compliance during test (unit: A)

Inputs	
`double Id_target=1e-4`	If not `nil`, Vtci is calculated and output instead of Vtex; enter positive value for NMOS and negative value for PMOS
`double Vd_sat=1.5`	`nil`: Do not measure Id_sat Double: measure Id_sat (unit: V)
`double Vd_leak=1.5`	`nil`: Do not measure Id_leak Double: Measure Id_leak under given Vd_leak (unit: V)
`double Vg_leak=1`	`nil`: Do not measure Ig_leak Double: Measure Ig_leak under given Vg_leak (unit: V)
`double Abort_shift=10 in [0,]`	When relative shift of parameters ((value[now] – value[fresh])/value[fresh]) reaches this value, abort (except Vt)
`double Abort_Vt=0.05 in [0,]`	When absolute shift of Vt (value[now] – value[fresh]) reaches this value, abort (unit: V)
`integer Abort_Ig=1000 in [0,]`	`nil`: Do not monitor on gate-current during stress Integer: When Ig[now]>=Ig[fresh]*Abort_Ig, abort
`double time_interval=1e-3 in [0,]`	Time interval between sampling of Ig if Ig is to be monitored during stress (unit: S)
`double Vg_stress=3`	Stress voltage on gate (unit: V)
`double Vd_stress=3.5`	Stress voltage on drain (unit: V)
`double Vb_stress=0`	Stress voltage on bulk (unit: V)
`double G_stress_comp=0.1 in [0,]`	Current limit on gate during stress (unit: A)
`double D_stress_comp=0.1 in [0,]`	Current limit on drain during stress (unit: A)

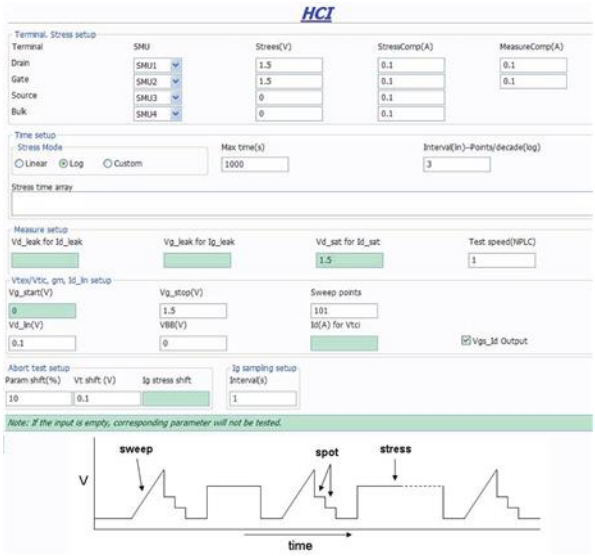
Intended outputs	
`'Time'`	Stress time section
`'Vtci', 'Vtex', 'gm', 'Id_lin', 'Id_sat', 'Id_leak', 'Ig_leak'`	Absolute value of measured parameters
`'Vtci_shift', 'Vtex_shift', 'gm_shift','Id_lin_shift', 'Id_sat_shift','Id_leak_shift' and 'Ig_leak_shift'`	Relative shift of measured parameter
`'Idi' and 'Vgi' (I = 1,2,3)`	Id_Vg curves
`'Ig' and 'Ig_time'`	Monitored gate leakage current and time during stress

Details

The following figure shows the user interface dialog box for HCI testing. If the imported Test Script Processor (TSP™) file has a corresponding `.xrc` user interface file, ACS automatically loads and displays the user interface.

Refer to the `.xrc` interface file for more information on importing `.xrc` files.

Figure 34: User interface for HCI



Terminal stress setup: Set the SMUs for each terminal, set the voltage and corresponding compliances during the stress and test. If the source or bulk is set to `KI_GND`, connect them to ground manually.

Time setup: Set the stress time. If Linear or Log is selected, leave the stress time array entry field blank. If Custom is selected, input the time array into the stress time array entry field.

Measure setup: Several tests are available. If a green test field remains empty, the corresponding test is not performed.

NOTE

If the Vg_start entry field is completed, but the Id(A) for Vtci entry field is empty, the threshold voltage (Vth) is extracted from the maximum gm. If the Id(A) for Vtci entry field is also completed, the Vth is extracted from the constant current.

Abort test setup: Set the parameters controlling the proceeding of the test. If the Ig stress shift entry field is completed, the gate-current Ig is monitored during stress, and if Ig[now] = Ig stress shift'*Ig[fresh], the test ends.

Pin connections: A four-terminal MOSFET is used in this test. The source and bulk can be connected to ground manually and two to four SMUs are needed. The process consists of two parts: test and stress.

For the stress, the stress time setting, linear/logarithmic/input-array, is set by you. In addition to the stress time, you can also monitor the gate-current during the stress test.

For the test, the following are supported:

- Threshold voltage 'Vtex' / 'Vtic', maximum conductance 'gm' and linear drain-current 'Id_lin' tests. If start gate-voltage 'Vg_start' is not empty, this test will be performed. If Id_target is not empty, Vtic (Vt extracted by constant current method) will be provided instead of Vtex (Vt extracted by maximum gm method).
- Saturate the drain-current 'Id_sat' test. If drain-voltage saturation 'Vd_sat' is not empty, measure the Id_sat@Vd=Vd_sat, and the Vg = Vd_sat.
- A drain leakage-current 'Id_leak' test. If the drain leakage-voltage 'Vd_leak' is not empty, measure the Id_leak@Vd = Vd_leak, and the Vg = Vb.
- A gate leakage-current 'Ig_leak' test. If the gate leakage voltage 'Vg_leak' is not empty, measure the Ig_leak@Vg=Vg_leak, and the Vd = Vs.

NOTE

The test will abort if a parameter exceeds its preset limit or the time limit (set by you) is completed.

Example

```
local VBB=0
local npdec_delta=4
local Abort_shift=50
local Vd_sat=nil
local B_comp=0.1
local t_max=20
local D_comp=0.1
local Vg_stop=1.4
local DSMU=SMU1
local t_mode=1
local time_input=nil
local G_stress_comp=0.1
local Id_Vg=1
local Vd_stress=0.5
local myNPLC=0.001
local time_interval=1
local D_stress_comp=0.1
local BSMU=KI_GND
local SSMU=KI_GND
local Vg_start=nil
local Vd_leak=2
local Vg_points=141
local Vd_lin=0.1
local Id_target=nil
local Abort_Ig=1000
local Vb_stress=0
local G_comp=0.1
local Vg_leak=2
local S_comp=0.1
local Abort_Vt=0.1
local VSS=0
local GSMU=SMU2
local Vg_stress=0.5
HCI(t_mode, t_max, npdec_delta, time_input, SSMU, BSMU, GSMU, DSMU, myNPLC, VSS,
    S_comp, VBB, B_comp, Id_Vg, Vg_start, Vg_stop, Vg_points, G_comp, Vd_lin,
    D_comp, Id_target, Vd_sat, Vd_leak, Vg_leak, Abort_shift, Abort_Vt, Abort_Ig,
    time_interval, Vg_stress, Vd_stress, Vb_stress, G_stress_comp, D_stress_comp)
```

Also see

None

TDDB_CCS

This test script performs the constant current time dependent dielectric breakdown (TDDB) test.

Usage

```
TDDB_CCS(sample_interval, time_max, holdtime, V_min, HBDL, myPLC, smu_1, comp1,
   stress1, meas1, smu_2, comp2, stress2, meas2, smu_3, comp3, stress3, meas3,
   smu_4, comp4, stress4, meas4)
```

Inputs	
`double sample_interval=1 in(0, )`	Time between sample (unit: s)
`double HBDL=0.6 in [0,0.999]`	Limit of hard breakdown (HBD); when Vg[now]≥Vg[prev]*HBDL, then abort
`double V_min=0.06 in [0,200]`	Minimum voltage
`double time_max=nil in(0, )/nil`	Maximum time of experiment; if `nil` appears, test until breakdown (BD)
`double holdtime=0 in[0, )`	Time before stress begins (unit: s)
`double myPLC=1 in[0.001,25]`	PLC setting
`integer smu_1=1 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`double comp1=2`	SMU compliance (unit: A for current; V for voltage)
`double stress1=1e-6`	SMU required stress value (unit: A for current; V for voltage)
`integer meas1=1 in [0,1]`	`1`: Current stress and make measurement `0`: Voltage stress no measurement
`integer smu_2=2 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`double comp2=0.1`	SMU compliance (unit: A for current; V for voltage)
`double stress2=0`	Stress value required on the SMU (unit: A for current; V for voltage)
`integer meas2=0 in [0,1]`	`1`: Current stress and make measurement `0`: Voltage stress no measurement
`integer smu_3=0 in[0,1,2..64]`	Maximum four SMUs are supported; if no SMUs, input `0`
`double comp3=nil`	SMU compliance (unit: A for current; V for voltage)
`double stress3=nil`	SMU required stress value (unit: A for current; V for voltage)
`integer meas3=nil`	`1`: Current stress and make measurement `0`: Voltage stress no measurement
`integer smu_4=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`double comp4=nil`	SMU compliance (unit: A for current; V for voltage)
`double stress4=nil`	SMU required stress value (Unit: A for current; V for voltage)
`integer meas4=nil`	`1`: Current stress and make measurement `0`: Voltage stress no measurement

Outputs	
`error`	Error message
`time1`	Time array of SMU1

Outputs	
`Vg1`	Voltage of SMU1
`TBD1`	Tbd of SMU1
`BD_type1`	Breakdown type of SMU1: `1` for HBD; `2` for timeout
`time2`	Time array of SMU2
`Vg2`	Voltage of SMU2
`TBD2`	Tbd of SMU2
`BD_type2`	Breakdown type of SMU2
`time3`	Time array of SMU3
`Vg3`	Voltage of SMU3
`TBD3`	Tbd of SMU3
`BD_type3`	Breakdown type of SMU3
`time4`	Time array of SMU4
`Vg4`	Voltage of SMU4
`TBD4`	Tbd of SMU4
`BD_type4`	Breakdown type of SMU4

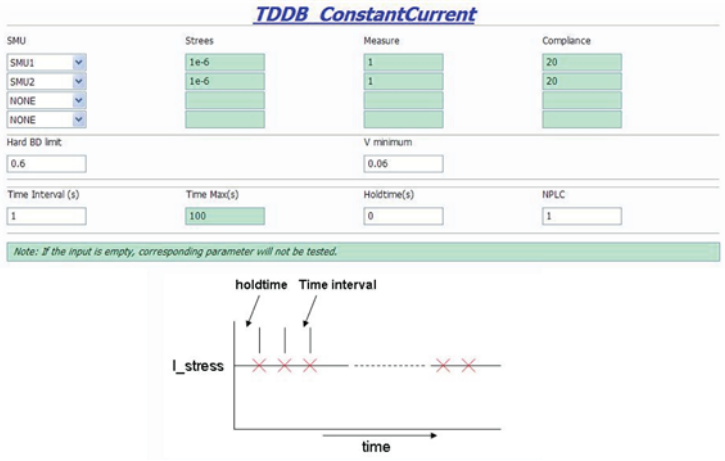
Details

Up to four SMUs are supported and only voltage is measured. The hard breakdown (HBD) occurs if:

- The Vg is below breakdown voltage (abs(Vg)<abs(Vmin))
- The Vg falls dramatically (abs(Vg[now]) ≤ HBDL * abs(Vg[prev]))

The following figure shows the dialog box for the `TDDB_CCS` test. A general description of this dialog box is included below.

Figure 35: User interface for TDDB_CCS



TDDB_CCS user interface descriptions

- **Terminal setting:** If the SMU is NONE, Stress, Measure and Compliance can be empty.
- **Measure:** Set the Measure column to `1` if you want to measure the SMU; set it to `0` if you only want to run a stress test.
- **Hard BD limit and V minimum:** Set the hard breakdown limit and voltage minimum. The unit is volts.
- **Time arrangement:** Time Max can be left empty. In this case, the test continues until all devices fail.

Example

```
local sample_interval=1
local time_max=50
local holdtime=0
local V_min = 0.06
local HBDL=0.6
local myPLC = 1
local smu_1=1
local comp1=20
local stress1=3e-6
local meas1=1
local smu_2=2
local comp2=0.1
local stress2=0
local meas2=0
local smu_3=0
local comp3=nil
local stress3=nil
local meas3=nil
local smu_4=0
local comp4=nil
local stress4=nil
local meas4=nil
TDDB_CCS(sample_interval, time_max, holdtime, V_min, HBDL, myPLC, smu_1, comp1,
    stress1, meas1, smu_2, comp2, stress2, meas2, smu_3, comp3, stress3, meas3,
    smu_4, comp4, stress4, meas4)
```

Also see

None

TDDB_per_pin

This test script performs a time-dependent dielectric breakdown (TDDB) test.

Usage

```
TDDB_per_pin(time_interval, HBDL, BD_mode, time_max, SBDL, AVL, holdtime, bas_num,
   SBD_num, smu1, comp1, stress1, meas1, smu2, comp2, stress2, meas2, smu3, comp3,
   stress3, meas3, smu4, comp4, stress4, meas4)
```

Inputs	
`double time_interval=0.01`	Time between sample (unit: s)
`integer HBDL=1000`	Limit of hard breakdown (HBD); when Ig[now]≥Ig[prev]*HBDL, then abort
`integer BD_mode=0`	`0`: HBD only; all the parameters related to soft breakdown (SBD) can be set to `nil` `1`: Also SBD
`double time_max=nil`	Maximum time of experiment; if `nil` appears, test until breakdown (BD)
`integer SBDL=500`	Limit of SBD; when Inoi[now] = Inoi[base]*SBDL, then abort
`integer AVL=10`	Standard when calculating base noise current; if Inoi[now]≤AVL*Inoi_base, Inoi [now] should be included in the Inoi_base calculation
`double holdtime=0`	Time before stress begins (unit: s)
`integer bas_num=6`	Number of noise current used to calculate Inoi_base value
`integer SBD_num=5`	Number of noise used to determine SBD
`integer smu1=1`	Maximum 4 SMUs are supported; if no SMUs, input `nil`
`double comp1=0.1`	SMU compliance (unit: A)
`double stress1=3`	SMU required stress value (unit: V)
`integer meas1=1`	`1`: Make measurement on this SMU `0`: No measurement
`integer smu2=2`	Maximum 4 SMUs are supported; if no SMUs, input `nil`
`double comp2=0.1`	SMU compliance (unit: A)
`double stress2=3`	SMU required stress value (unit: V)
`integer meas2=1`	`1`: Make measurement on this SMU `0`: No measurement
`integer smu3=0`	Maximum 4 SMUs are supported; if no SMUs, input `nil`
`double comp3=nil`	SMU compliance (unit: A)
`double stress3=nil`	SMU required stress value (unit: V)
`integer meas3=nil`	`1`: Make measurement on this SMU `0`: No measurement
`integer smu4=0`	Maximum 4 SMUs are supported; if no SMUs, input `nil`
`double comp4=nil`	SMU compliance (unit: A)
`double stress4=nil`	Stress value required on the SMU (unit: V)
`integer meas4=nil`	`1`: Make measurement on this SMU `0`: No measurement

Details

Up to four SMUs are supported, voltage is forced, and current is measured.

If the breakdown mode is `0`, hard breakdown (HBD) is monitored. If the breakdown mode is `1`, soft breakdown (SBD) is also monitored.

HBD occurs when Ig[now] = HBDL*Ig[prev].

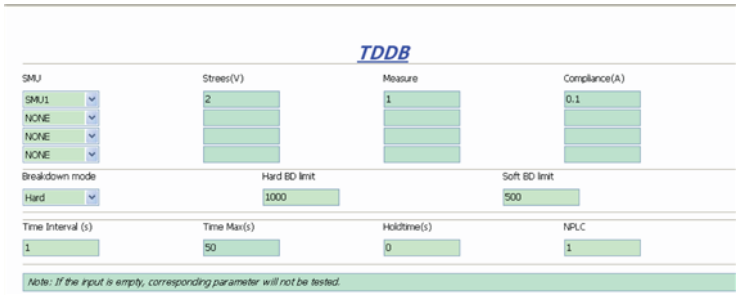
To evaluate the SBD, calculate the noise of the gate current (Inoi) from the formula listed in the JEDEC standard JESD92, available at jedec.org. The base noise (`Inoi_base`) is calculated with the Inoi average value (AVL) and base number (`bas_num`). When `Inoi_base` is set, SBD occurs if several sequential `Inois` conditions are met: `Inoi [now] = SBDL*Inoi_base`.

If the device under test (DUT) is a MOSFET, set the SMUs that do not need to measure to `0` (meas=0).

Intended outputs: Time, Ig, Ig_noise (when SBD is required), and breakdown_type of SMUs requiring measurement.

The following figure shows the dialog box for the TDDB test. A general description of this dialog box is included below.

Figure 36: User interface for TDDB



TDDB_CCS GUI descriptions

- **Terminal setting:** If the SMU is set to NONE in the SMU list, you need to set the Measure(V), Breakdown settings, and the Time arrangement.
- **Measure(V):** Set the Measure(V) column to `1` if you want to measure the SMU; set to `0` if you only want to run a stress test.
- **Breakdown mode:** If Breakdown mode is set to Hard; Soft BD can be left empty.
- **Time arrangement:** Time Max can be left empty; the test continues until all devices fail.

Example

```
local time_interval=0.005
local HBDL=1000
local BD_mode=0
local time_max=20
local SBDL=500
local AVL=10
local holdtime=0
local bas_num=6
local SBD_num=5
local smu1=1
local comp1=0.1
local stress1=2
local meas1=1
local smu2=0
local comp2=nil
local stress2=nil
local meas2=nil
local smu3=0
local comp3=nil
local stress3=nil
local meas3=nil
local smu4=2
local comp4=0.1
local stress4=2
local meas4=1
TDDB_per_pin(time_interval, HBDL, BD_mode, time_max, SBDL, AVL, holdtime,
    bas_num, SBD_num, smu1, comp1, stress1, meas1, smu2, comp2, stress2, meas2,
    smu3, comp3, stress3, meas3, smu4, comp4, stress4, meas4)
```

Also see

None

NBTI

This test script runs a negative bias temperature instability (NBTI) test.

Usage

```
NBTI(smu_D, smu_G, smu_S, smu_B, vg_stress, vd_stress, vg_meas, vd_meas, myNPLC,
   meas_delay, navg, t_array, modeflag, compliancei, time, did)
```

Inputs	
instid smu_D=SMU1	SMU1, SMU2, SMU3, ..., SMU64
instid smu_G=SMU2	SMU1, SMU2, SMU3, ..., SMU64
instid smu_S=KI_GND	SMU1, SMU2, SMU3, ..., SMU64, KI_GND
instid smu_B=KI_GND	SMU1, SMU2, SMU3, ..., SMU64, KI_GND
double vg_stress=-2.0 in [-40,40]	Gate stress voltage
double vd_stress=0 in [-40,40]	Drain stress voltage
double vg_meas=-1.2 in [-40,40]	Gate measure voltage
double vd_meas=-1.2 in [-40,40]	Drain measure voltage
double myNPLC=0.001 in [0.001,25]	NPLC, 0.001 to ~ 10
double meas_delay=0 in [0,]	Measure delay after stress is off
integer navg=1 in [1,20]	Double of points for average
table t_array={1,2,5,10,20,50,100}	Stress time array
integer modeflag=1 in [0,1]	Gate first or drain first
double compliancei=0.1 in [0,]	Current compliance

Outputs	
time={}	Timetable
did={}	Drain-current shift table

Details

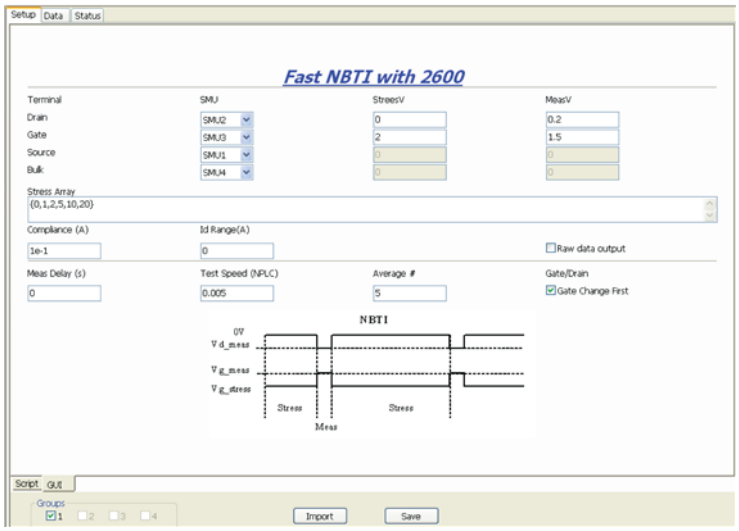
The NBTI test supports two to four SMUs.

The gate performs the stress test and the drain performs the measurement test. In most cases, the source and bulk are set to 0 or KI_GND.

Intended outputs: Time, id0 (fresh value of drain-current), id (absolute value of drain-current), and id_shift (relative shift of drain-current).

The following figure shows the NBTI dialog box and illustrates the testing method. A general description of this dialog box is included below.

Figure 37: User interface for NBTI



NBTI user interface descriptions

- **Terminal settings:** SMUs are assigned to terminals source and bulk and KI_GND is manually set. The voltage is changeable only on the gate and drain. A measurement is made on the drain only, and compliance should be set.
- **Test Speed setting entry field:** The Meas Delay field sets the time before each measurement. The Test Speed field sets the PLC value. The Average # field specifies the number of measurements on which the average is taken.
- **Gate/Drain:** Voltages are applied on the gate and drain and change when the measurement begins and ends. The gate/drain selection box specifies which terminal changes first. If Gate Change First is selected, the gate terminal changes first. If the Gate Change First is cleared, the drain terminal changes first.
- **Stress Array:** Used to input the time array.

Example

```
local compliancei=1e-1
local modeflag=0
local vd_meas=0.1
local navg=1
local t_array={0,1,2,5,10,20}
local smu_B=SMU4
local smu_D=SMU2
local smu_G=SMU3
local myNPLC=0.01
local vg_meas=1.5
local meas_delay=0
local smu_S=SMU1
local vd_stress=0
local vg_stress=2
local time={}
local did={}
NBTI(smu_D, smu_G, smu_S, smu_B, vg_stress, vd_stress, vg_meas, vd_meas, myNPLC,
   meas_delay, navg, t_array, modeflag, compliancei, time, did)
```

Also see

None

NBTI_meas

This test script runs negative bias temperature instability (NBTI) tests with pre-Id_Vg testing and post-Id_Vg testing.

Usage

```
NBTI_meas(smuD, smuG, smuS, smuB, flag0, flag1, flag2, p_Vg_lo, p_Vg_hi,
   p_Vg_points, p_Vds, p_Drangei, p_sweepdelay, a, b, A, W, L, Vg_ini, Vd_ini,
   Vg_stress, Vd_stress, Vg_meas, Vd_meas, myNPLC, meas_delay, inter_delay, t_mode,
   t_max, npdec_delta, time_input, modeflag, Gcompi, Dcompi, rng, Nsam)
```

Inputs	
`instid smuD=SMU2`	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`
`instid smuG=SMU1`	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`
`instid smuS=KI_GND`	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`, `KI_GND`
`instid smuB=KI_GND`	`SMU1`, `SMU2`, `SMU3`, ..., `SMU64`, `KI_GND`
`integer flag0=1 in [0,1]`	Flag for idvg test `1`: Enable pre-idvg test and post-idvg test `0`: Disable the test
`integer flag1=1 in [0,1]`	Flag for NBTI test `1`: Enable NBTI stress-measure test `0`: Disable the test
`integer flag2=1 in [0,1]`	Flag for Vcti test `1`: meas enable Vcti test `0`: Disable the test
`double p_Vg_lo=0 in [-40, 40]`	Start of gate-voltage sweep in pre- and post-test

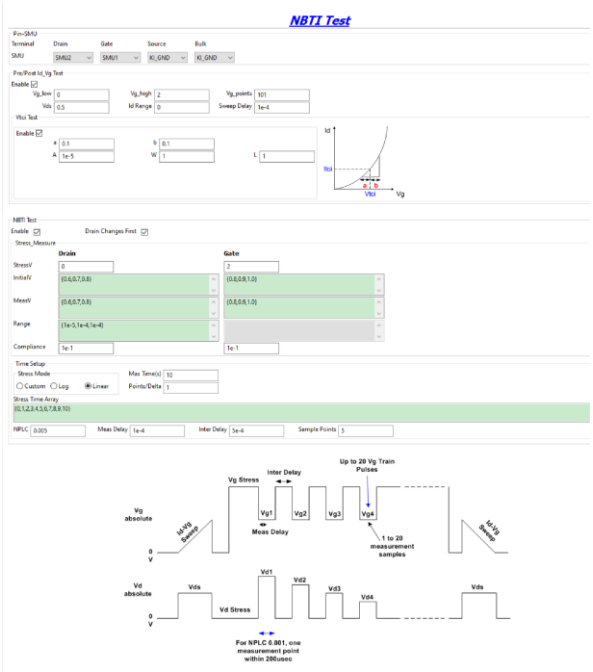
Inputs	
`double p_Vg_hi=2 in [-40, 40]`	Stop of gate-voltage sweep in pre- and post-test
`double p_Vg_points=21 in [0, 4096]`	Gate-voltage sweep number of points in pre- and post-test
`double p_Vds=1  in [-40, 40]`	Drain-source bias in pre- and post-test
`double p_Drangei=1e-3 in [0, 0.1]`	Drain-current range in pre- and post-test
`double p_sweepdelay=0 in [0,]`	Sweep delay in pre- and post-test
`double a=0 in [0,40]`	Low extent of Vtci sweep
`double b=1 in [0,40]`	High extent of Vtci sweep
`double A=1 in [0,]`	Target current density
`double W=1 in [0,]`	Width of device
`double L=1 in [0,]`	Length of device
`table Vg_ini in [-40, 40]`	Gate-voltage for initial drain-current measurement
`table Vd_ini in [-40, 40]`	drain-voltage for initial drain-current measurement
`double Vg_stress=-2.0 in [-40, 40]`	Gate stress voltage
`double Vd_stress=0 in [-40, 40]`	Drain stress voltage
`table Vg_meas in [-40, 40]`	Gate measure voltage
`table Vd_meas in [-40, 40]`	Drain measure voltage
`double myNPLC=0.05 in [0.001, 25]`	NPLC, `0.001` to `25`
`double meas_delay=0.001 in [0,]`	Measure delay after stress is off
`double inter_delay=0.1 in [0,]`	Delay between measure voltage train pulses
`integer t_mode=1 in [0,2]`	`0`: User-specified time array `1`: Logarithmic time `2`: Linear time array
`double t_max=20 in [0,]`	The maximum stress time; valid when t_mode is `1` or `2`
`double npdec_delta in [0,]`	When t_mode = `1`: Number of points-per-decade When t_mode = `2`: Delta time
`table time_input in [0,]`	When t_mode is `0`, this array is used as a stress time list
`integer modeflag=1 in [0, 1]`	Measurement force gate first or drain first modeflag=`0`: Drain first modeflag=`1`: Gate first
`double Gcompi = 100e-6 in [0, 0.1]`	Gate-voltage source compliance
`double Dcompi = 100e-6 in [0, 0.1]`	Drain-voltage source compliance
`table rng in [0, 0.1]`	Drain-current measure range
`integer Nsam = 5 in [1, 20]`	Number of samples

Outputs	
error	Working condition flag
Vg_pre	Gate voltage of pretest
Id_pre	Drain-current of pretest
Vg_pos	Gate-voltage of post-test
Id_pos	Drain-current of post-test
Vtci	Gate voltage at target drain current
Idini	Initial drain current
Idend	Drain-current after stress sequence
time	Timetable
Id1	Drain current table
Id2	
Id3	
Id4	
Id5	
Id6	
Id7	
Id8	
Id9	
Id10	
Id11	
Id12	
Id13	
Id14	
Id15	
Id16	
Id17	
Id18	
Id19	
Id20	

Details

The following figure shows the NBTI_meas test dialog box and illustrates the testing method.

Figure 38: User interface for NBTI_meas



Example

```
local p_sweepdelay=1e-4
local Vd_ini={0.6,0.7,0.8}
local modeflag=1
local p_Vds=0.5
local Nsam=5
local npdec_delta=1
local meas_delay=1e-4
local t_max=10
local Dcompi=1e-1
local t_mode=2
local Gcompi=1e-1
local time_input={0,1,2,3,4,5,6,7,8,9,10}
local inter_delay=5e-4
local flag2=1
local flag1=1
local flag0=1
local Vd_stress=0
local myNPLC=0.005
local A=1e-5
local Vg_ini={0.8,0.9,1.0}
local rng={1e-5,1e-4,1e-4}
local L=1
local p_Drangei=0
local p_Vg_hi=2
local W=1
local p_Vg_points=101
local p_Vg_lo=0
local a=0.1
local b=0.1
local Vd_meas={0.6,0.7,0.8}
local smuS=KI_GND
local smuB=KI_GND
local Vg_meas={0.8,0.9,1.0}
local smuG=SMU1
local smuD=SMU2
local Vg_stress=2
NBTI_meas(smuD,smuG,smuS,smuB,flag0,flag1,flag2,p_Vg_lo,p_Vg_hi,p_Vg_points,p_Vds
    ,p_Drangei,p_sweepdelay,a,b,A,W,L,Vg_ini,Vd_ini,Vg_stress,Vd_stress,Vg_meas,Vd
    _meas,myNPL,meas_delay,inter_delay,t_mode,t_max,npdec_delta,time_input,modefla
    g,Gcompi,Dcompi,rng,Nsam)
```

Also see

None

NBTI_on_the_fly

This test script monitors threshold voltage degradation and relaxation for NBTI and charge trapping on high K gate stacks.

Usage

```
NBTI_on_the_fly(Test_mode, Vg_stress, Vg_relax, Vg_dist, Vd, Stress_time,
   Monitor_time_stamp, GSMU, DSMU, SSMU, BSMU, myNPLC)
```

Inputs	
`integer Test_mode=2`	`0`: Monitor Vt degradation during stress only `1`: Monitor Vt relaxation during stress off only `2`: Monitor both degradation and relaxation during stress on and off period
`double Vg_stress=3`	Voltage on gate during stress; measurement during stress is also made at this voltage
`double Vg_relax=1`	Measure voltage on gate during recovery; the stress voltage recovery is set to `0`
`double Vg_dist=0.05`	Delta Vg for different Id measurement
`double Vd=0.1`	Drain voltage only applied during monitoring; other times `0` V
`integer Stress_time=1000`	Time for stress in seconds
`table Monitor_time_stamp={}`	Time in seconds; this is an input array that monitors the guiding time for two instances; the actual timestamp for monitoring might not be the same due to measurement time; also, this timestamp is the same for both stress on (degradation monitoring) and off (relaxation monitoring)
`instid GSMU=SMU1`	Gate SMU number, for example, SMU1
`instid DSMU=SMU2`	Drain SMU number, SMU2
`instid SSMU=KI_GND`	Source SMU number; if KI_GND is chosen, this terminal should be connected to GNDU manually
`instid BSMU=KI_GND`	Bulk SMU number
`double myNPLC=0.01`	NPLC setting

Outputs	
`'ERROR' (possible error type)`	`-1`: Wrong inputs
`'Time_stress', 'dVt_stress' and 'Id_stress'`	Time, Vt shift, and drain current during stress phase
`'Time_relax', 'dVt_ relax' and 'Id_ relax'`	Time, Vt shift, and drain current during relax phase

Details

This script is a method for monitoring threshold voltage degradation and relaxation for NBTI and charge trapping on high K gate stacks. The code is a Keithley Instruments copyright.

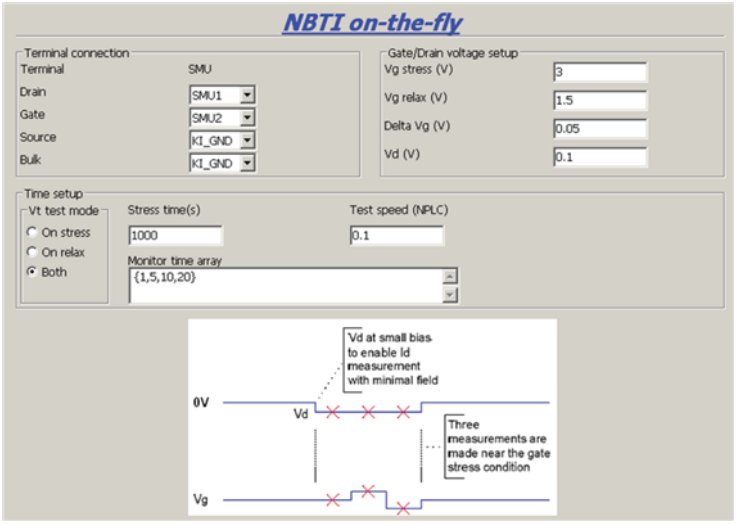
- `Vg_stress` is for stress and measurement during the stress phase.
- `Vg_relax` is for measurement during recovery.

- Recovery voltage is set to 0 during no measurement times.
- This test can only be used for one device, once during the stress-on period and once during the stress-off period.

Reference: "On-the-fly characterization of NBTI in ultra-thin gate oxide PMOSFETs," M. Denais, et. al, IEDM 2004.

The following figure shows the NBTI_on_the_fly test dialog box and illustrates the testing method. A general description of this dialog box is included below.

Figure 39: User interface for NBTI_on_the_fly



NBTI user interface descriptions

- **Terminal connection:** SMUs are assigned to source and bulk terminals and KI_GND is manually set. If specific SMUs are assigned to these two terminals, 0 V is applied internally.
- **Gate/Drain voltage setup:** The voltage during the stress phase and relax phase on the gate and drain should be set here.
- **Time setup:** Specifies time during the stress and relaxation phases. For the Vt test mode, when On stress is selected, there is no relax phase and stress is applied following the monitor time array. If On relax is selected, a measurement is made during the relax phase only following the monitor time array, and stress time is specified by Stress time. If both are selected, a measurement is made during both the stress phase and the relax phase, and they both follow monitor time array.

Example

```
local Test_mode = 2
local Vg_stress = 3
local Vg_relax = 1.5
local Vg_dist = 0.05
local Vd = 0.1
local Stress_time = 1000
local Monitor_time_stamp = {1,5,10,20}
local GSMU = SMU2
local DSMU = SMU1
local SSMU = KI_GND
local BSMU = KI_GND
local myNPLC = 0.1
NBTI_on_the_fly(Test_mode, Vg_stress, Vg_relax, Vg_dist, Vd, Stress_time,
    Monitor_time_stamp, GSMU, DSMU, SSMU, BSMU, myNPLC)
```

Also see

None

QBD_rmpj

This test script performs a charge-to-breakdown test using the QBD Ramp J test algorithm described in the JEDEC standard JESD35-A, *Procedure for Wafer-level Testing of Thin Dielectrics*, available at jedec.org.

Usage

```
function qbd_rmpj(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use, I_init,
    I_start, F, t_step, exit_volt_mult, V_max, I_max, q_max, area)
```

Inputs	
`integer HiSMUId=1 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId3=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId2=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId3=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`double myplc=1 in[0.001,25]`	PLC setting
`double v_use=1 in[-200,200]`	Oxide voltage under normal operating conditions (V); typically the power-supply voltage of the process; this voltage is to measure pre- and post-voltage ramp oxide current
`double I_init=1e-5 in[-0.1,0.1]`	Oxide breakdown failure current when biased at v_use (A); typical value is 10 ΩA/cm^2 and may change depending on oxide area; for maximum sensitivity, the specified value should be well above the worst-case oxide current of a good oxide and well above system noise floor; higher value must be specified for ultra-thin oxide because of direct tunneling effect
`double I_start=1e-5 in[-0.1,0.1]`	Starting current for current ramp (A); typical value is I_init
`double F=1.5 in[1,100]`	Current multiplier between two successive current steps
`double t_step=0.1 in(0,)`	Current ramp step time in seconds

Inputs	
`double exit_volt_mult=0.85 in[0,2]`	Multiplier factor of successive voltage measurement; when the next measured voltage is below this factor multiplying previous measured voltage, oxide is considered breakdown and test exits; typical value: 0.85
`double V_max=20 in[-200,200]`	The voltage limit; pay attention to interlock (A)
`double I_max=0.1 in[-0.1,0.1]`	Maximum ramp up current (A)
`double q_max=100 in(0,)`	Maximum accumulated oxide charge per oxide area (C/cm^2); terminates a test where breakdown occurs but was not detected during the test Note that the maximum accumulated oxide charge is *per oxide area*.
`double area=2 in(0,)`	Area of oxide structure (cm^2)

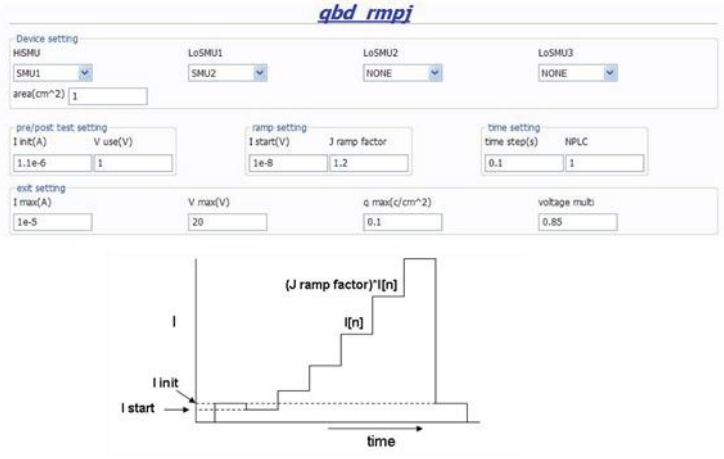
Outputs	
`V_stress`	Voltage array
`I_stress`	Timestamp array representing when current is measured
`q_stress`	Accumulated charge array per oxide area
`V_init_pre`	Voltage at I_init in pretest
`V_init_post`	Voltage at I_init in post-test
`Q_bd`	Charge to breakdown; cumulative charge passing through the oxide prior to breakdown (C)
`q_bd`	Charge to breakdown density (C/cm^2)
`v_bd`	Applied voltage at the step just before oxide breakdown
`I_bd`	Measured current at v_bd just before oxide breakdown
`t_bd`	Timestamp when measuring I_bd
`Failure_mode`	`1`: Initial test failure `2`: Catastrophic failure (initial test pass, ramp test fail, post-test fail) `3`: Masked catastrophic (initial test pass, ramp test pass, post-test fail) `4`: Noncatastrophic (initial test pass, ramp test fail, post-test pass) `5`: Others (initial test pass, ramp test pass, posttest pass)
`Test_status`	`0`: No test errors (exit due to measured voltage < exit_volt_mult*V_previous) (`-1`): Failed prestress test (`-2`): Cumulative charge limit reached (`-3`): Current limit reached (`-4`): Voltage limit reached (`-5`): Masked catastrophic failure (`-6`): Noncatastrophic failure (`-7`): Invalid specified t_step

Details

This algorithm forces a logarithmic current ramp until the oxide layer breaks down. This algorithm can produce a maximum current of ±1 A if using a high-power source-measure unit (SMU).

The following figure shows the QBD Ramp J dialog box and illustrates the testing method.

Figure 40: User interface for qbd-rmpj



NOTE

If the above routine is modified, change the function name to avoid possible programming errors.

Example

```
local HiSMUId=1
local LoSMUId1=2
local LoSMUId2=0
local LoSMUId3=0
local myplc=1
local v_use=0.005
local I_init=1e-8
local I_start=1e-8
local F=1.5
local t_step=0.1
local exit_volt_mult=0.85
local V_max=20
local I_max=1e-5
local q_max=0.1
local area=1
qbd_rmpj(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use, I_init, I_start, F,
    t_step, exit_volt_mult, V_max, I_max, q_max, area).
```

Also see

None

QBD_rmpv

This test script performs a charge-to-breakdown test using the QBD Ramp V Test algorithm described in the JEDEC standard JESD35-A, *Procedure for Wafer-level Testing of Thin Dielectrics*, published April 2001.

Usage

```
qbd_rmpv(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use, I_init, hold_time,
   v_start, v_step, t_step, measure_delay, I_crit, I_box, I_max, exit_curr_mult,
   exit_slope_mult, q_max, t_max, v_max, area, exit_mode)
```

Inputs	
`integer HiSMUId=1 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId1=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId2=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`integer LoSMUId3=0 in[0,1,2..64]`	Maximum 4 SMUs are supported; if no SMUs, input `0`
`double myplc=1 in[0.001,25]`	PLC setting
`double v_use=1 in[-200,200]`	Oxide voltage under normal operating conditions (V); typically the power-supply voltage of the process; this voltage is to measure pre- and post-voltage ramp oxide current
`double I_init=0.001 in[-0.1,0.1]`	Oxide breakdown failure current when biased at v_use (A); typical value is 10 ΩA/cm^2 and may change depending on oxide area; for maximum sensitivity, the specified value should be well above the worst-case oxide current of a good oxide and well above system noise floor; higher value must be specified for ultra-thin oxide because of direct tunneling effect
`double holdtime=0 in[0, ]`	Time after `v_use` is applied (unit: s)
`double v_start=0.01 in[-200,200]`	Starting ramp voltage (V); typical value is `v_use`
`double v_step=0.01 in[-200,200]`	Voltage ramp step size (V); this value has a maximum value of 0.1 MV/cm; for example, the maximum value can be calculated using T_{ox}*0.1MV/cm, where T_{ox} is in centimeters; this is 0.1 V for a 10 nm oxide
`double t_step=0.1 in(0,)`	Voltage ramp step time (unit: s); determines the voltage ramp rate; this time should be less or equal than 100 ms (typically 40 ms to 100 ms)
`double measure_delay=0.05 in(0,)`	Time delay for measurement after each voltage stress step (unit: s); this delay should be less than `t_step`
`double I_crit=5e-4 in[-0.1,0.1]`	At least 10 times the test system current measurement noise floor; this oxide current is the minimum value used in determining the change of slope breakdown criteria (A)
`double I_box=3e-4 in[-0.1,0.1]`	An optional measured current level for which a stress voltage is recorded; this value provides an additional point on the current-voltage curve; typical value is 1 µA
`double I_max=1e-3 in[-0.1,0.1]`	Oxide breakdown criteria; `I_bd` is obtained from I-V curves and is the oxide current at the step just before breakdown

Inputs	
`double exit_curr_mult=10 in(0,)`	Change of current failure criteria; this is the ratio of measured current over previous current level, which, if exceeded, results in failure; recommended value: 10 to 100
`double exit_slope_mult=3 in(0,)`	Change of slope failure criteria; this is the factor of change in FN slope, which, if exceeded, results in failure; recommended value: 3
`double q_max=100 in(0,)`	Maximum accumulated oxide charge per oxide area (C/cm^2); terminates a test where breakdown occurs but was not detected during the test Note that the maximum accumulated oxide charge is *per oxide area*.
`double t_max=10 in(0,)`	Maximum stress time allowed (unit: s); reaching the limit results in test finish
`double v_max=2 in(-200,200)`	The maximum voltage limit for the voltage ramp; this limit is specified at 30 MV/cm for oxides less than 20 nm thick and 15 MV/cm for thicker oxides; for example, `v_max` can be estimated from T_{ox}*30 MV/cm, where T_{ox} is in centimeters; this is 35 V for a 10.0 nm oxide
`double area=2 in(0,)`	Area of oxide structure (cm^2)
`integer exit_mode=0 in(0,1)`	Failure criteria mode: ■ `0`: Judge by current (`I_max`) and (`exit_curr_mult`) and `q_max`, `v_max`, `t_max` ■ `1`: Also judge slope (`exit_slope_mult`)

Inputs	
`V_stress`	Voltage array
`I_stress`	Current array
`T_stress`	Timestamp array representing when current is measured
`q_stress`	Accumulated charge array per oxide area
`I_use_pre`	Measured oxide current at `v_use` before starting the ramp
`Q_bd`	Charge to breakdown; cumulative charge passing through the oxide prior to breakdown (C)
`q_bd`	Charge to breakdown density (C/cm^2)
`v_bd`	Applied voltage at the step just before oxide breakdown
`I_bd`	Measured current at v_bd just before oxide breakdown
`t_bd`	Timestamp when measuring I_bd
`v_crit`	Applied voltage at the step when the oxide current exceeds `I_crit`
`v_box`	Applied voltage at the step when the oxide current exceeds `I_box`

Inputs	
`Failure_mode`	`1`: Initial test failure `2`: Catastrophic failure (initial test pass, ramp test fail, post-test fail) `3`: Masked catastrophic (initial test pass, ramp test pass, post-test fail) `4`: Noncatastrophic (initial test pass, ramp test fail, post-test pass) `5`: Others (initial test pass, ramp test pass, posttest pass)
`Test_status`	`0`: No test errors (exit due to measured voltage < exit_volt_mult*V_previous) (`-1`): Failed prestress test (`-2`): Cumulative charge limit reached (`-3`): Current limit reached (`-4`): Voltage limit reached (`-5`): Masked catastrophic failure (`-6`): Noncatastrophic failure (`-7`): Invalid specified t_step

Details

This algorithm forces a linear voltage ramp until the oxide layer breaks down. This algorithm can produce a maximum voltage of ±200 V.

The following figure shows the QBD Ramp V dialog box and illustrates the testing method.

Figure 41: User interface for qbd_rmpv

Example

```
local HiSMUId=1
local LoSMUId1=2
local LoSMUId2=0
local LoSMUId3=0
local myplc=1
local v_use=1
local I_init=0.001
local hold_time=0
local v_start=0.01
local v_step=0.01
local t_step=0.1
local measure_delay=0.05
local I_crit=5e-4
local I_box=3e-4
local I_max=1e-3
local exit_curr_mult=10
local exit_slope_mult=3
local q_max=100
local t_max=100
local v_max=2
local area=2
local exit_mode=1
qbd_rmpv(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use, I_init, hold_time,
   v_start, v_step, t_step, measure_delay, I_crit, I_box, I_max, exit_curr_mult,
   exit_slope_mult, q_max, t_max, v_max, area, exit_mode)
```

Also see

None

Em_iso_test

This test script runs isothermal electromigration tests that are compliant with the JEDEC standard JESD 61 (October 2007), available at jedec.org.

Usage

```
RunEmIsoTestEngine (smun, test_mode, width, thickness, n_wide, n_narrow, TCR, Tref,
   Vlimit, Ilimit, init_failR, Tinit, start_J,step_J, step_delay, equil_time,
   Ttarget, Terror, Rfail_fact, max_time, max_count)
```

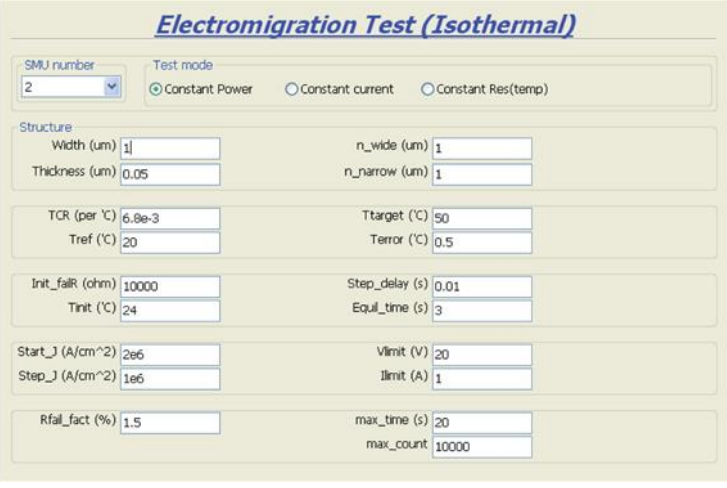
Inputs	
`integer smun = 4`	Number of SMUs (`1` to `4`)
`integer test_mode = 0`	`0`: Constant power `1`: Constant current `2`: Constant resistance (temp)
`double width = 1`	Width of the structure (microns)
`double thickness = 0.05`	Thickness of the structure (microns)
`double n_wide = 1`	Number of wide squares in the structure (use `1` for straight-line structure)
`double n_narrow = 1`	Number of narrow squares in the structure (use `1` for straight-line structure)
`double TCR = 6.8e-3`	Temperature coefficient of resistance at Tref (per °C)
`double Tref = 20`	Reference temperature for TCR (°C)
`double Vlimit = 40`	Voltage limit (V)
`double Ilimit = 1`	Current limit (A)
`double init_failR`	Room temperature failure resistance (Ω)
`double Tinit = 24`	Initial temperature of the device (°C)
`double start_J = 2e6`	Starting current density in A/cm2 (typical: 1e6 A/cm^2)
`double step_J = 1e6`	Current density step in A/cm2
`double step_delay = 0.01`	Delay after source and before measure (typical: 50 ms to 100 ms)
`double equil_time = 3`	Maximum time for convergence control loop (s)
`double Ttarget = 300`	Target temperature of the metal line (°C)
`double Terror = 0.5`	Error band allowed to control temperature (°C)
`double Rfail_fact =1.5`	Percent resistance increase for device failure (applies only in constant stress mode)
`double max_time = 60`	Maximum test time (s)
`double max_count = 10000`	Maximum data points to measure

Outputs	
Time_smu1	Timestamp for device under test (DUT) 1 (s)
source_smu1	Source value for DUT 1 (A)
Reading_smu1	Readings for DUT 1 (Ω)
Temp_smu1	Temperature for DUT 1 (K)
Time_smu2	Timestamp for DUT 2 (s)
source_smu2	Source value for DUT 2 (A)
Reading_smu2	Readings for DUT 2 (Ω)
Temp_smu2	Readings for DUT 2 (Ω)
Time_smu3	Timestamp for DUT 3 (s)
source_smu3	Source value for DUT 3 (A)
Reading_smu3	Readings for DUT 3 (Ω)
Temp_smu3	Temperature for DUT 3 (K)
Time_smu4	Timestamp for DUT 4 (s)
source_smu4	Source value for DUT 4 (A)
Reading_smu4	Readings for DUT 4 (Ω)
Temp_smu4	Temperature for DUT 4 (K)

Details

The following figure shows the user interface for the Electromigration Test (Isothermal).

Figure 42: User interface for ISO_EM test



Example

```
local n_narrow=1
local Ttarget=50
local test_mode=0
local Ilimit=1
local n_wide=1
local width=1
local step_delay=0.01
local init_failR=10000
local TCR=6.8e-3
local Rfail_fact=1.5
local thickness=0.05
local step_J=1e6
local smun=2
local max_count=10000
local max_time=20
local Terror=0.5
local start_J=2e6
local Tref=20
local Vlimit=20
local equil_time=3
local Tinit=24

RunEmIsoTestEngine(smun, test_mode, width, thickness, n_wide, n_narrow, TCR,
   Tref, Vlimit, Ilimit, init_failR, Tinit, start_J, step_J, step_delay,
   equil_time, Ttarget, Terror, Rfail_fact, max_time, max_count)
```

Also see

None

DMM7500 and DMM6500 library

This section provides example digital multimeter (DMM) example test module applications using the Keithley DMM7500 and DMM6500 and Keithley source-measure units (SMUs). These applications include:

- Configure an LIV library to measure photodiode current (on page 4-66)
- Configure a DMM_SMU_lib library to measure metal line or low resistance on 2- or 3-terminal devices (on page 4-69)

Configure a LIV library to measure photodiode current

The following topics describe how to configure an LIV library with a Keithley Instruments Model 2610B-PULSE SourceMeter and a DMM7510 or DMM6500. Light output, current, and voltage (LIV) characterization is an important and commonly used way to verify the performance of all types of optical devices, including light-emitting diodes (LEDs), laser diodes, vertical cavity surface emitting lasers (VCSELs), and edge emitting lasers (EELs).

This example creates a module that calls a single function to output a current pulse sweep with the Model 2601B-PULSE current pulser. The sweep is generated with the Asynchronous Trigger Model of the instrument. With the dual 1 MS/s digitizers built into the current pulser, the voltage and current are measured simultaneously at the top of each pulse, and photodiode (PD) current is measured by the DMM7510 or DMM6500. Upon completion of the sweep, the tested data is printed to the Test Script Builder instrument console in a format suitable for copying and pasting into Microsoft® Excel® for graphing and analysis.

This library is based on *Application Note: Timing Considerations for Constructing LIV Measurement Trigger Models Using the Keithley 2601B-PULSE System SourceMeter® Instrument and DMM7510 Graphical Sampling Digital Multimeter*, which can be accessed on tek.com (tek.com/en).

Set up a measurementLIV_CaseI_2601B_PULSE_DMM test module

The following is an example of setting up a test module named `measurementLIV_CaseI_2601B_PULSE_DMM`.

To add a measurementLIV_CaseI_2601B_PULSE_DMM test module:

1. Add a script test module (STM) to the configuration navigator.
2. Import the `LIV.tsp` module from the TSPLib library.
3. Select the `measurementLIV_CaseI_2601B_PULSE_DMM` module from test module drop-down list.

Required equipment:

- Model 2601B-PULSE System SourceMeter
- DMM7510 or DMM6500 Multimeter

NOTE

The function does not perform any error checking. You are responsible for specifying settings that are compatible with the instrument model being used and with the power envelope of that instrument.

⚠ WARNING

It is the user's responsibility to follow all safety guidelines given in the user manuals for each instrument you use. This is especially critical if voltages exceeding 42 VDC are present in the test circuit. Such voltage level is hazardous and could cause personal injury or death.

The following table describes the input parameters for this example.

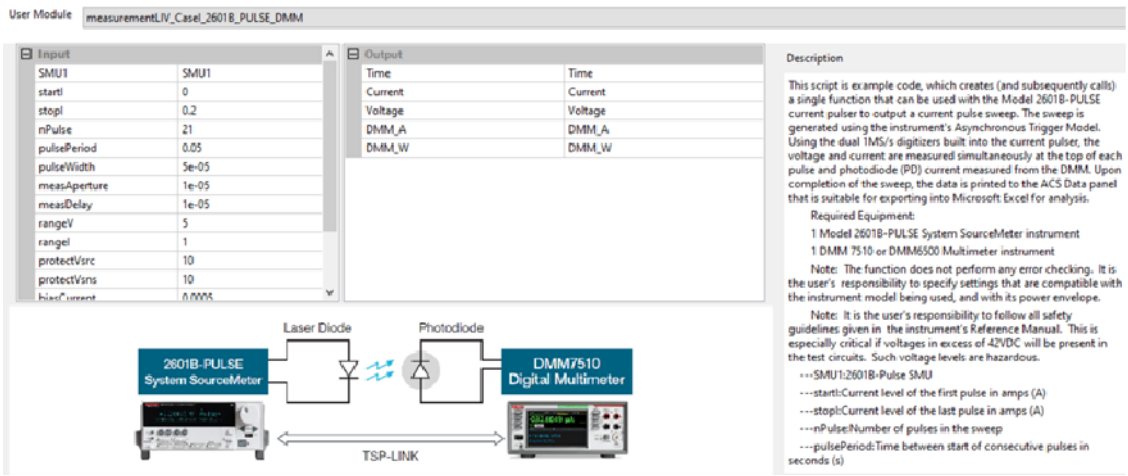
Input	Description
SRC_SMU	2601B-Pulse SMU for source
startI	Current level of the first pulse (A)
stopI	Current level of the last pulse (A)
nPulse	Number of pulses in the sweep
pulsePeriod	Time between start of consecutive pulses (s)
pulseWidth	Width of current pulses (s)
measAperture	Effective integration time (s)
measDelay	Time from pulse start to measure start (s)
rangeV	Voltage measure range (V)
rangeI	Current source and measure range (A)
protectVsrc	Protection Voltage on Source side (V)
protectVsns	Protection Voltage on Sense side (V)
biasCurrent	Idle current level (base level for pulses) (A)
rangeDMM	Range of DMM (A)
optoFactor	Proportional Constant to calculate optical power based on the photodiode current
printData	1: print test data in the Data panel.

The following table describes the outputs for this example.

Output	Description
Time	Time stamp (s)
Current	Current reading (A)
Voltage	Voltage reading (V)
DMM_A	DMM measured current (A)
DMM_W	DMM measured power (W)

The following figure shows the `measurementLIV_CaseI_2601B_PULSE_DMM` module in ACS.

Figure 43: Standard user interface for the measurementLIV_CaseI_2601B_PULSE_DMM module



Configure a DMM_SMU_lib library to measure metal line or low resistance on 2- or 3-terminal devices

The following examples describe how to configure a library that measures metal line or low resistance with low voltage on 2-terminal or 3-terminal devices using a Keithley Instruments 2400 Series SourceMeter instrument or Series 2600 SourceMeter instrument as the source and a DMM7510 or DMM6500 to make the measurements.

Set up a DMM_SMU_FIMV_Sample test module

This example module sources current using a Series 2400 TTI SourceMeter or Series 2600 SourceMeter and measures voltage using a DMM7510 or DMM6500 Multimeter using the specified sample points and delay time. It then outputs the test data.

To add a DMM_SMU_FIMV_Sample Test module:

1. Add a script test module (STM) to the configuration navigator.
2. Import the `DMM_SMU_lib.tsp` module from the TSPLib library.
3. Choose the `DMM_SMU_FIMV_Sample` module from the test module drop-down list.

Required equipment:

- 2400 Graphical Series SourceMeter instrument or Series 2600 SourceMeter instrument
- DMM7510 or DMM6500 Multimeter

The following table describes the input parameters for this example.

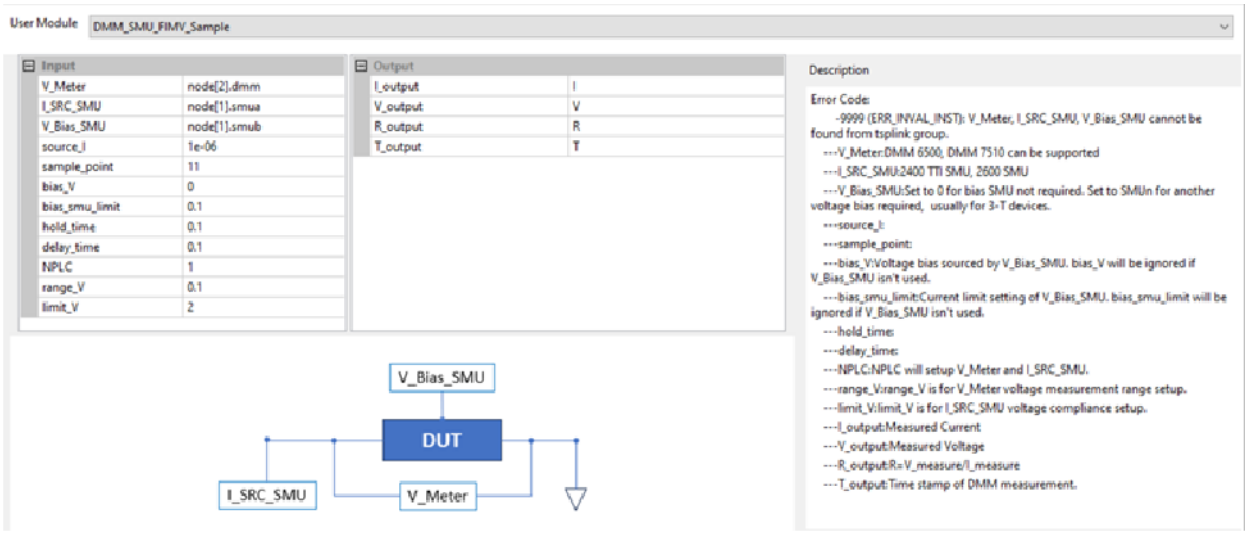
Input	Description
V_Meter	DMM7510 or DMM6500
I_SRC_SMU	2400 TTI SMU or 2600 SMU
V_Bias_SMU	Set to 0 if bias SMU is not required. Set to SMU*n* (2400 TTI SMU or 2600 SMU) if bias SMU is required, usually for 3-terminal devices.
source_I	Source current (A)
sample_point	Sample point. Set to 1 for spot measurement.
bias_V	Voltage bias sourced by *V_Bias_SMU*. (V) Will be ignored if bias SMU is not required and V_Bias_SMU is set to 0.
hold_time	Hold time (s)
delay_time	Delay time (s)
NPLC	Number of power line cycles. Will be applied for *V_Meter* and *I_SRC_SMU*.
range_V	Voltage measure range for *V_Meter*. (V)
limit_V	Voltage compliance for *I_SRC_SMU*. (V)

The following table describes the outputs for this example.

Output	Description
Error	-9999 (ERR_INVAL_INST): V_Meter, I_SRC_SMU, V_Bias_SMU cannot be found in TSP-Link group
I_output	Output current (A)
V_output	Output voltage (V)
R_output	Output resistor (ohm)

The following figure shows the DMM_SMU_FIMV_Sample test module in ACS.

Figure 44: Standard user interface for the DMM_SMU_FIMV_Sample module



Set up a DMM_SMU_FIMV_Sweep test module

To add a DMM_SMU_FIMV_Sweep Test module:

1. Add a script test module (STM) to the configuration navigator.
2. Import the `DMM_SMU_lib.tsp` module from the TSPLib library.
3. Select the `DMM_SMU_FIMV_Sweep` module from test module drop-down list.

Required equipment:

1. 2400 Graphical Series SourceMeter instrument or Series 2600 SourceMeter instrument
2. DMM7510 or DMM6500 Multimeter

This module sources current as a sweep using a 2400 Graphical Series SourceMeter instrument or Series 2600 SourceMeter instrument and measures voltage using a DMM7510 or DMM6500 Multimeter. It then outputs the test data.

The following table describes the input parameters for this example.

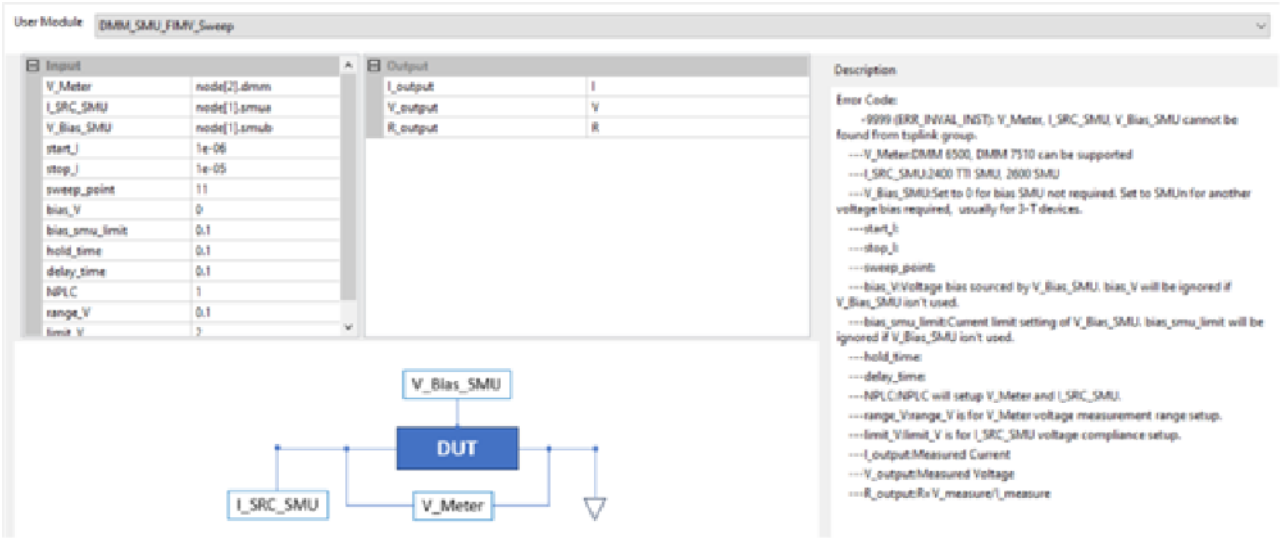
Input	Description
V_Meter	DMM7510 or DMM6500
I_SRC_SMU	2400 Series SMU or 2600B SMU
V_Bias_SMU	Set to 0 if bias, a SMU is not required (set the 2400 Series or 2600B SMU as SMU*X* if bias SMU is required; usually for 3-terminal devices)
start_I	Start the current sweep
stop_I	Stop the current sweep
sweep_point	Sweep point
bias_V	Voltage bias sourced by *V_Bias_SMU* (V) (ignored if bias SMU is not required and *V_Bias_SMU* is set to 0)
hold_time	Hold time (s)
delay_time	Delay time (s)
NPLC	Number of pulse line cycles. (applicable for *V_Meter* and *I_SRC_SMU*)
range_V	Voltage measurement range for *V_Meter* (V)
limit_V	Voltage compliance for *I_SRC_SMU* (V)

The following table provides the output for this example

Output	Description
Error	-9999 (ERR_INVAL_INST): V_Meter, I_SRC_SMU, V_Bias_SMU not found in TSP-Link group
I_output	Output current (A)
V_output	Output voltage (V)
R_output	Output resistor (ohm)

The following graphic shows the DMM_SMU_FIMV_Sweep test module in ACS.

Figure 45: Standard user interface for the DMM_SMU_FIMV_Sweep test module



VthSiC_JEP library

The following information helps you to configure a VthSiC_JEP library with Keithley Instruments Model 2600B SourceMeter. The VthSiC_JEP library provides modules to measure threshold voltage (Vth) for SiC devices based on the test principle as determined by the JEDEC standards (jedec.org) JEP183.

The threshold voltage (VT) is a key parameter for evaluating SiC MOSFETs. Without accurately measuring the threshold voltage, it is impossible to monitor how the stress applied to the device changes the device characteristics.

There are four modules in the library based on different test methods and sequence to measure VT for SiC MOSFETs:

1. VgSweepVdFixed (on page 4-74)
2. VgDualSweepVdFixed (on page 4-77)
3. VgVdBothSweep (on page 4-79)
4. VgVdBothFixedBias (on page 4-83)

VgSweepVdFixed

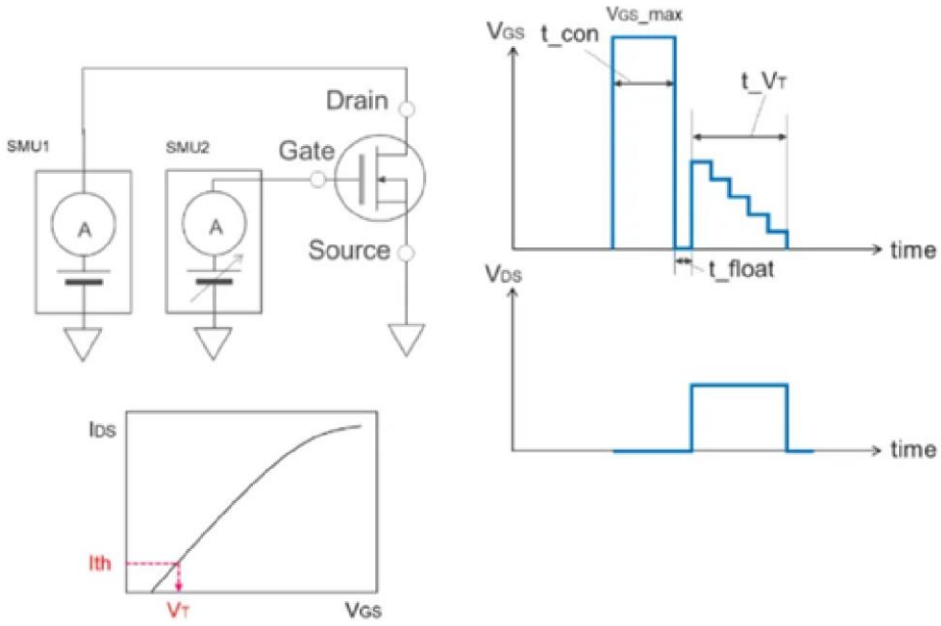
To add a VgSweepVdFixed test module:

1. Add a script test module (STM) to the configuration navigator.
2. Import the `VthSiC_JEP183.tsp` module from the TSPLib library.
3. Select the `VgSweepVdFixed` module from test module drop-down list.

Required equipment: Keithley Instruments Series 2600B System SourceMeter(R) instrument.

This module applies a positive Gate pulse (conditioning pulse) prior to measuring the Vth to eliminate the hysteresis (refer to JEDEC standards (on page 4-73) JEP183 for more information). Next, the module performs the GateV and DrainI measurement by sweeping the Gate voltage with a downward staircase sweep. The measured data of GateV and DrainI will be used to calculate the Vth. The following figure shows the Vth test circuit and timing diagram.

Figure 46: VgSweepVdFixed test circuit timing diagram



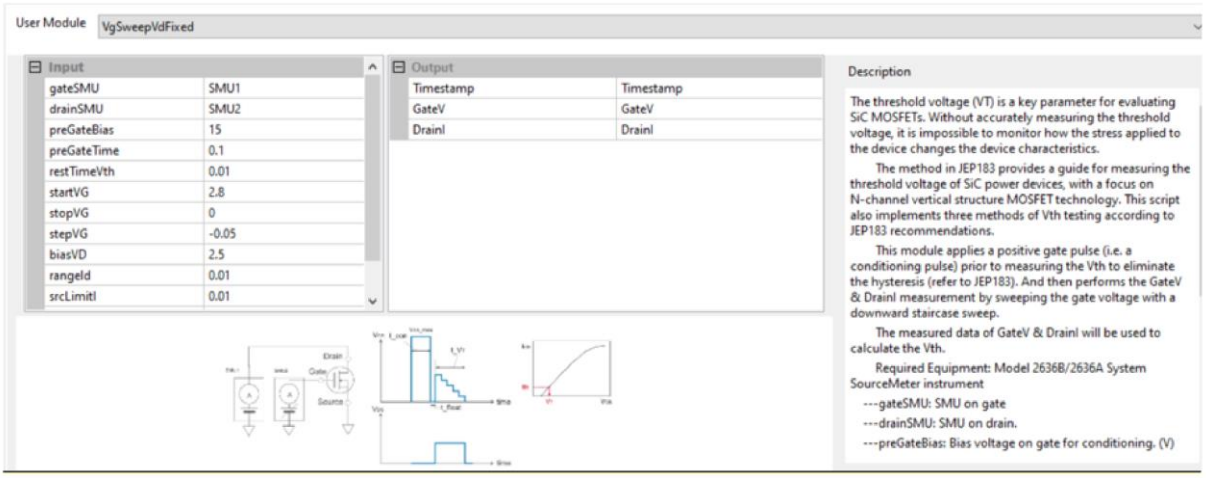
The following table provides the input and the description of the instrument testing when performing the `VgSweepVdFixed` sample test.

Input	Description
gateSMU	SMU on Gate
drainSMU	SMU on Drain
preGateBias	Bias voltage on Gate for conditioning (V)
preGateTime	Time for conditioning (s)
restTimeVth	Rest time after conditioning before Vth measurement (s)
startVG	Start voltage of sweep on Gate (V)
stopVG	Stop voltage of sweep on Gate (V)
stepVG	Step voltage of sweep on Gate (V)
biasVD	Bias voltage on drain (V)
rangeId	Current range on drain (A)
srcLimitI	Source current limit (A)
nPLC	Number of power line cycle
sweepDelay	Delay time for each step of sweep before measurement (s)

The following table provides the output and the description of the instrument testing when performing the `VgSweepVdFixed` sample test.

Output	Description
`Timestamp`	Time stamp reading. (s)
`GateV`	Gate voltage reading. (V)
`DrainI`	Drain current reading. (A)

Figure 47: ACSBasic_VgSweepVdFixed_GUI



You can calculate the VT with a formula and test data. By adding a formula VTLINGM(DrainI, GateV) in the formulator, VT can be calculated with linear GM, as shown in the following two figures.

Figure 48: Calculate the VT with a formula

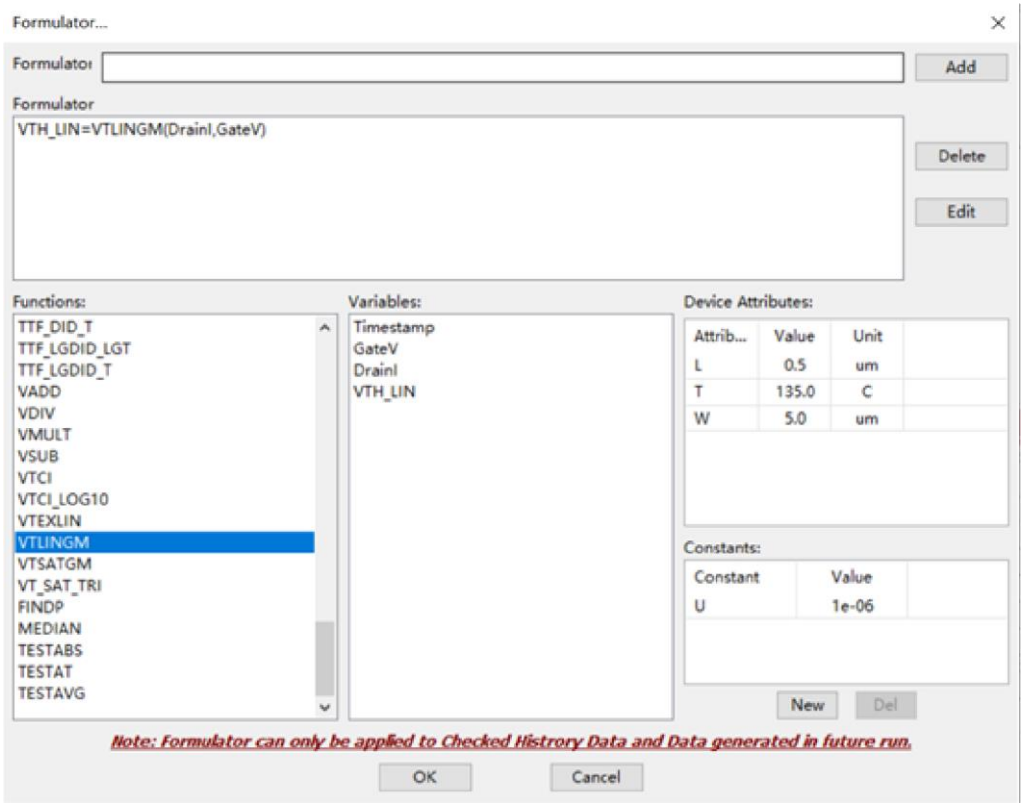
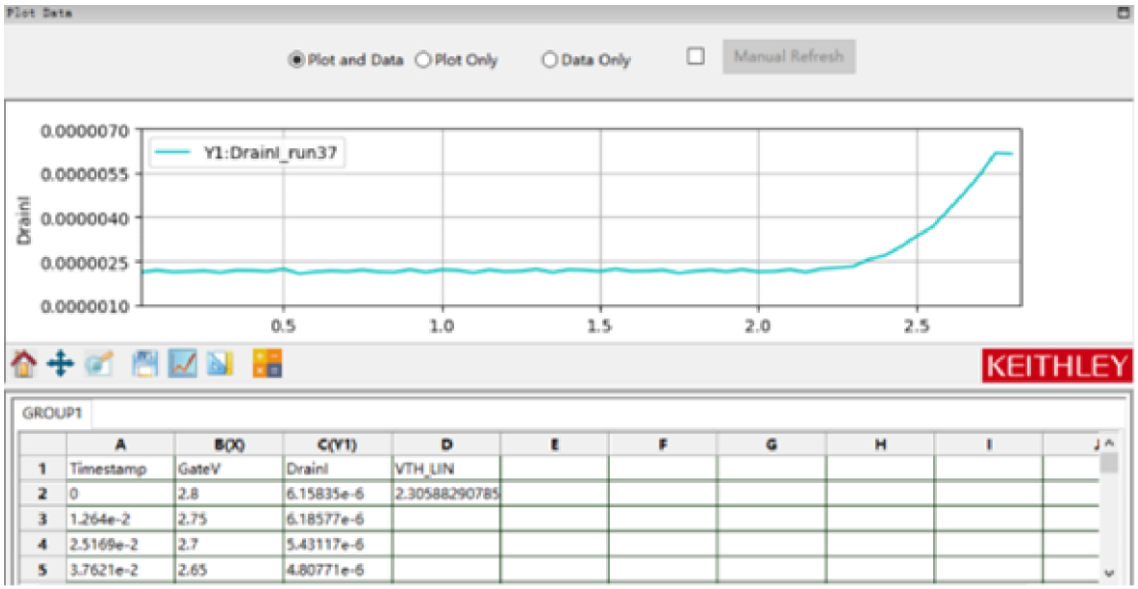


Figure 49: Plot data of the VGSweepVdFixed test



	A	B(X)	C(Y1)	D	E	F	G	H	I
1	Timestamp	GateV	DrainI	VTH_LIN					
2	0	2.8	6.15835e-6	2.30588290785					
3	1.264e-2	2.75	6.18577e-6						
4	2.5169e-2	2.7	5.43117e-6						
5	3.7621e-2	2.65	4.80771e-6						

VgDualSweepVdFixed test

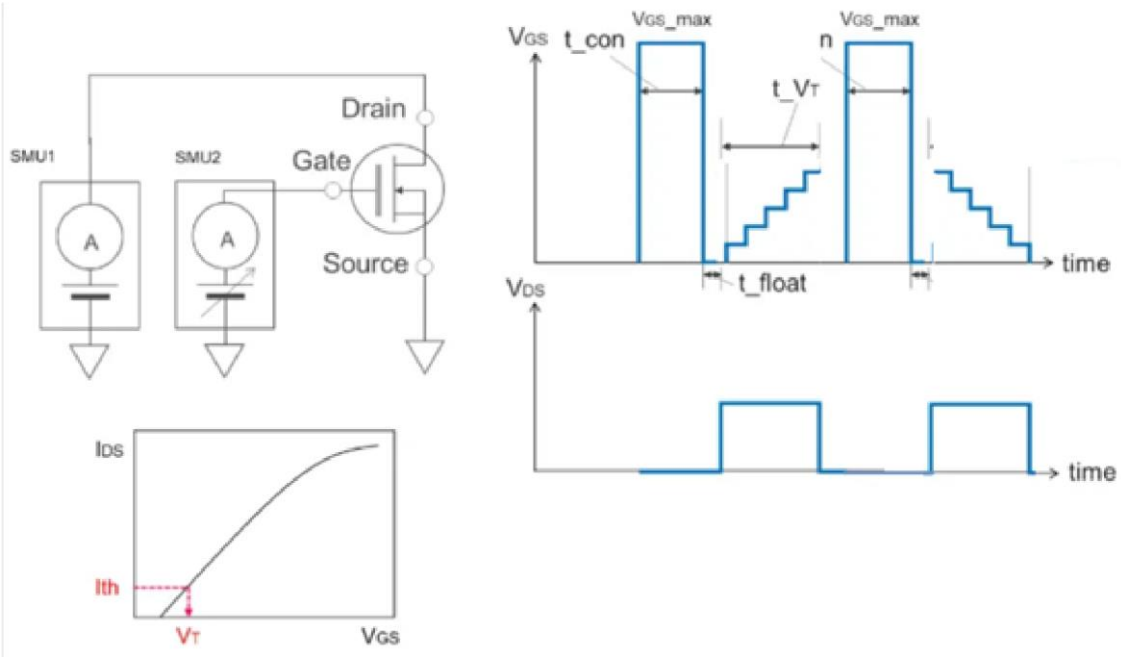
To add a VgDualSweepVdFixed test module:

1. Add a STM to the configuration navigator.
2. Import the `VthSiC_JEP183.tsp` module from the TSPLib library.
3. Select the `VgDualSweepVdFixed` module from test module drop-down list.

Required equipment: Series 2600 SourceMeter

This module applies a positive gate pulse prior to measuring the Vth to eliminate the hysteresis (refer to JEDEC standards (on page 4-73) JEP183 for more information). Next, it performs the GateV and DrainI measurement by dual sweeping the gate voltage with an upward staircase sweep and a downward staircase sweep. The measured data of GateV and DrainI will be used to calculate the Vth. The following figure shows the Vth test circuit and timing diagram.

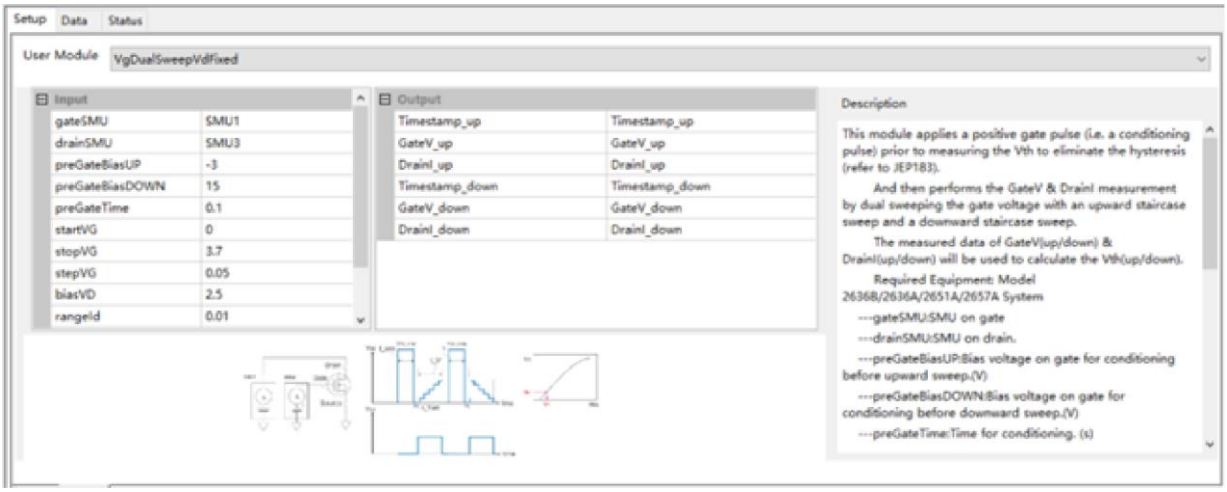
Figure 50: Vg dual sweep Vd fixed test circuit and timing diagram



Input	Description
gateSMU	SMU on Gate.
drainSMU	SMU on Drain.
preGateBiasUP	Bias voltage on gate for conditioning before upward sweep..(V)
preGateBiasDOWN	Bias voltage on gate for conditioning before downward sweep.(V)
preGateTime	Time for conditioning.(s)
restTimeVth	Rest time after pre-measurement before Vth measurement.(s)
startVG	Start voltage of sweep on Gate. (V)
stopVG	Stop voltage of sweep on Gate. (V)
stepVG	Step voltage of sweep on Gate. (V)
biasVD	Bias voltage on drain. (V)
rangeId	Current range on drain. (V)
srcLimitI	Source current limit. (V)
nPLC	Number of power line cycle. (V)
sweepDelay	Idle current level (base level for pulses) (A)

Output	Description
Time_up	Time stamp of upward sweep. (s)
GateV_up	Gate voltage reading of upward sweep. (V)
DrainI_up	Drain current reading of upward sweep. (A)
Time_down	Time stamp of downward sweep. (s)
GateV_down	Gate voltage reading of downward sweep. (V)
DrainI_down	Drain current reading of downward sweep. (A)

Figure 51: Vg dual sweep Vd fixed standard GUI



You can calculate the VT using a formula and test data. Refer to the Figure 361 for more information regarding how to calculate the VT using the VTLINGM formula.

VgVdBothSweep

To add a* VgVdBothSweep *test module:

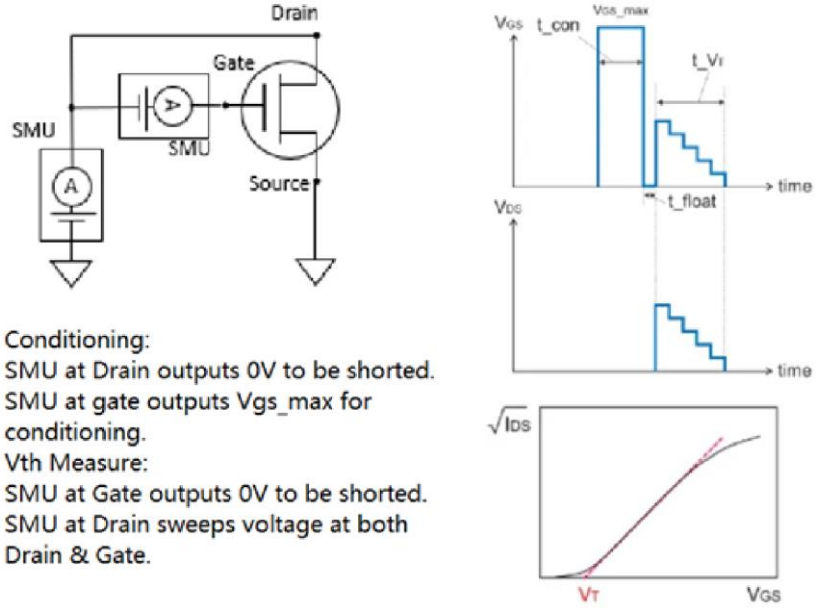
1. Add a script test module (STM) to the configuration navigator.
2. Import the `VthSiC_JEP183.tsp` module from the TSPLib library.
3. Select the `VgVdBothSweep` module from test module drop-down list.

Required Equipment: Series 2600 SourceMeter

This module applies a positive Gate pulse (conditioning pulse) prior to measuring the Vth to eliminate the hysteresis (refer to JEDEC standards (on page 4-73) JEP183 for more information). And then performs the GateV and DrainI measurement by sweeping the voltage on both the gate and drain synchronously with a downward staircase sweep. The measured data of GateV and DrainI are be used to calculate the Vth.

The following figure shows the Vth test circuit and timing diagram. Note that the SMU gate and SMU drain need to be connected in series, for instance the SMU-Drain is connected to HI, and the SMU-Gate is connected to LO.

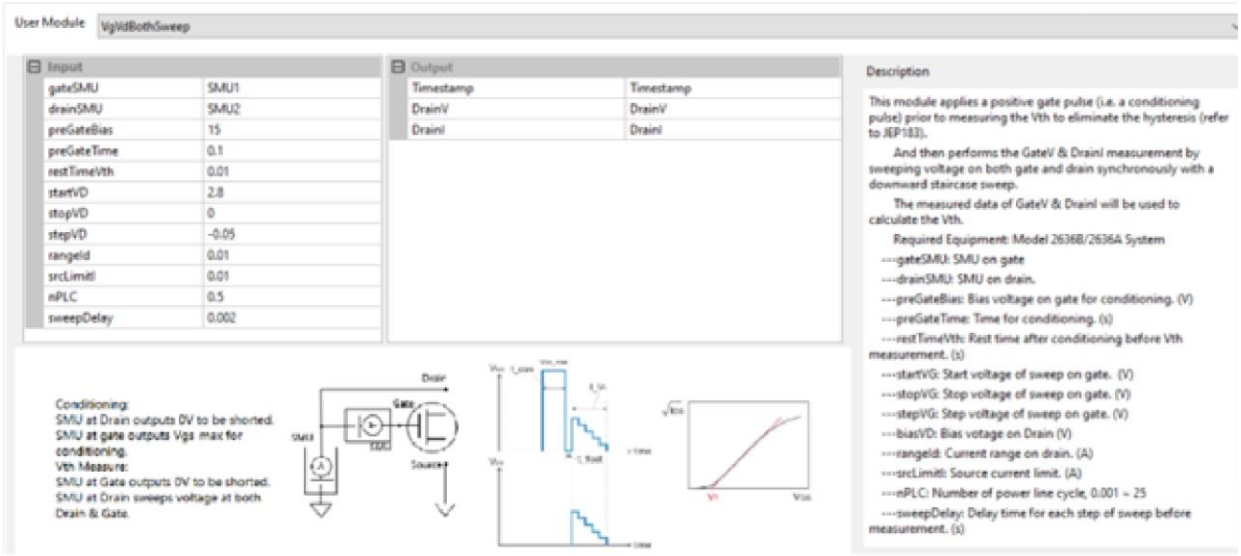
Figure 52: Vg Vd both sweep test circuit and timing diagram



Input	Description
gateSMU	SMU on gate.
drainSMU	SMU on drain.
preGateBias	Bias voltage on gate for conditioning.(V)
preGateTime	Time for conditioning.(s)
restTimeVth	Rest time after conditioning before Vth measurement.(s)
startVG	Start voltage of sweep on gate. (V)
stopVG	Stop voltage of sweep on gate. (V)
stepVG	Step voltage of sweep on gate. (V)
biasVD	Bias voltage on drain. (V)
rangeId	Current range on drain. (A)
srcLimitI	Source current limit. (A)
nPLC	Number of power line cycle.
sweepDelay	Delay time for each step of sweep before measurement. (s)

Output	Description
Timestamp	Time stamp reading. (s)
GateV	Gate voltage reading. (V)
DrainI	Drain current reading. (A)

Figure 53: Vg and Vd sweep standard GUI



You can calculate the VT with formula and test data by adding the formula VTSATGM (GateV and DrainI) in the formulator. VT is calculated using the GM saturation, as shown in the following figures.

Figure 54: Calculate VT with the VTSATGM formula

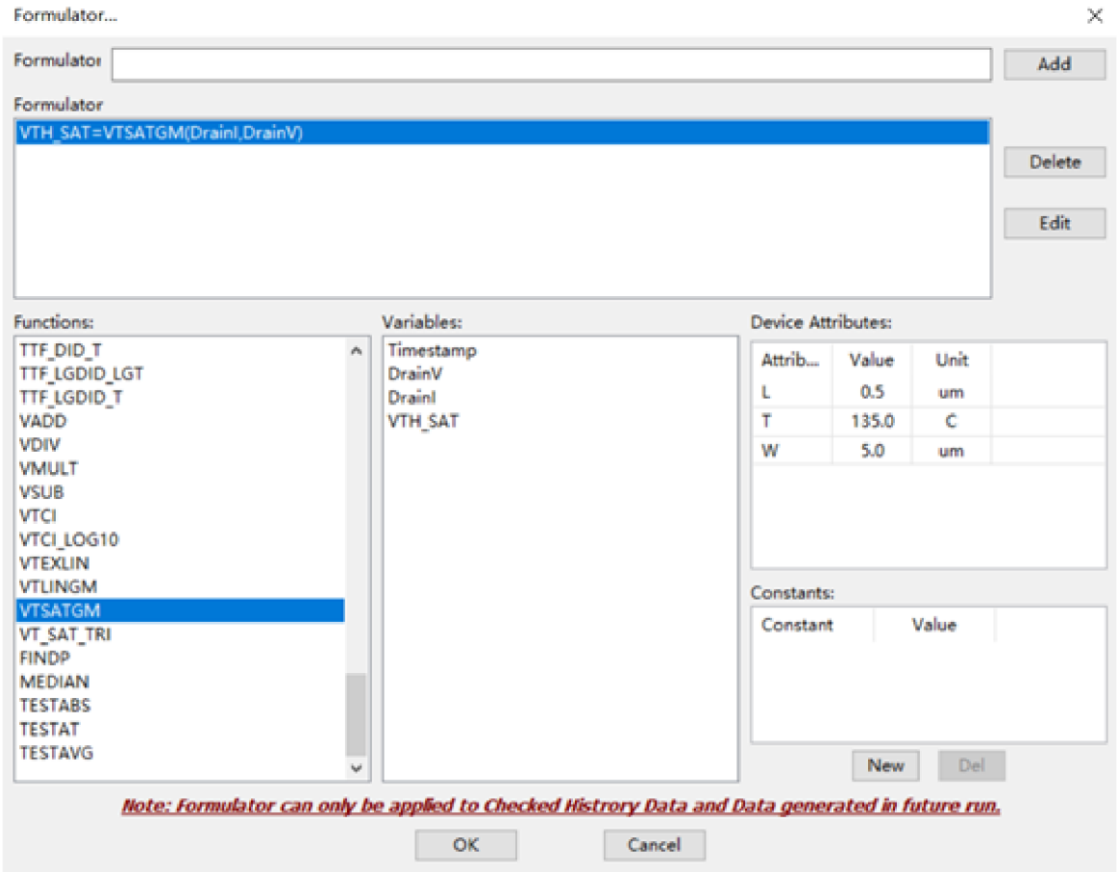
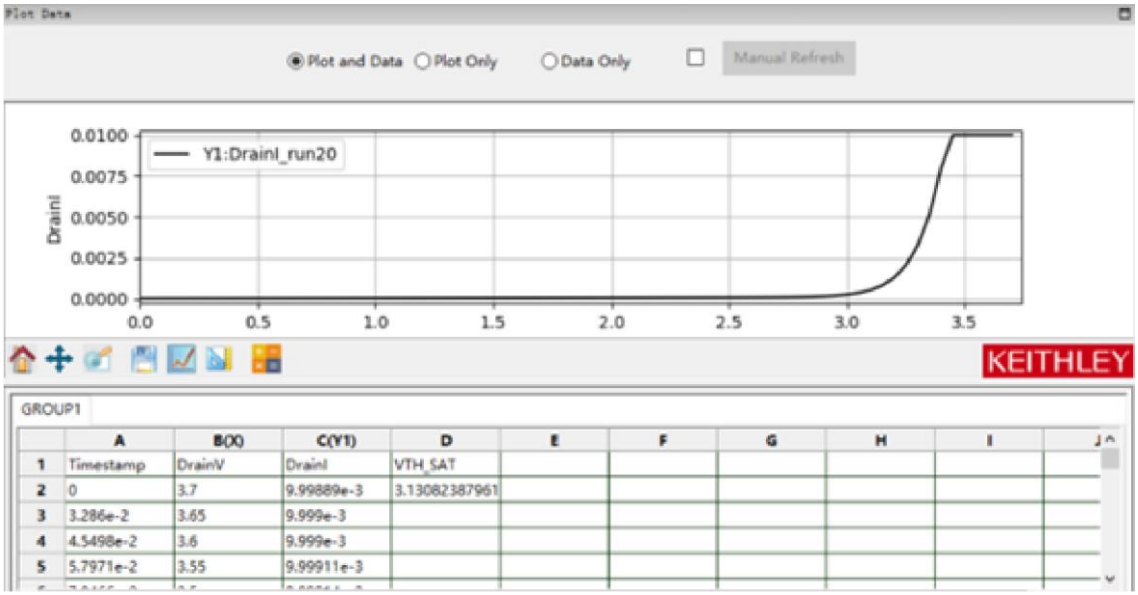


Figure 55: Plot Data for the VTSATGM



VgVdBothFixedBias

To add a VgVdBothFixedBias module:

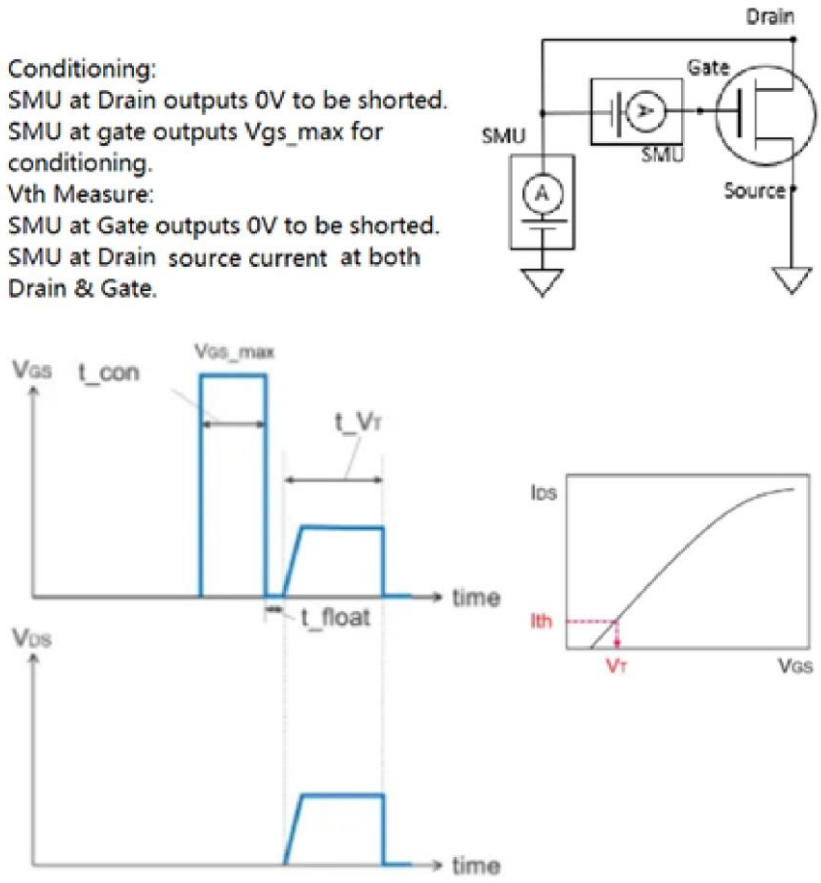
1. Add a STM to the configuration navigator.
2. Import the `VthSiC_JEP183.tsp` module from the TSPLib library.
3. Select the `VgVdBothFixedBias` module from test module drop-down list.

Required equipment: Series 2600 SourceMeter

This module applies a positive gate pulse prior to measuring the *Vth* to eliminate the hysteresis (refer to JEP183). And then sources a threshold current as *biasID* on the drain and gate at the same time, next it performs the gate voltage measurement. The measured gate voltage is the aimed threshold voltage *Vth*.

The following figure shows the *Vth* test circuit and timing diagram. Note that SMU on the gate and the SMU on drain must be connected in series (HI of the SMU-Drain connected to LO of the SMU-Gate).

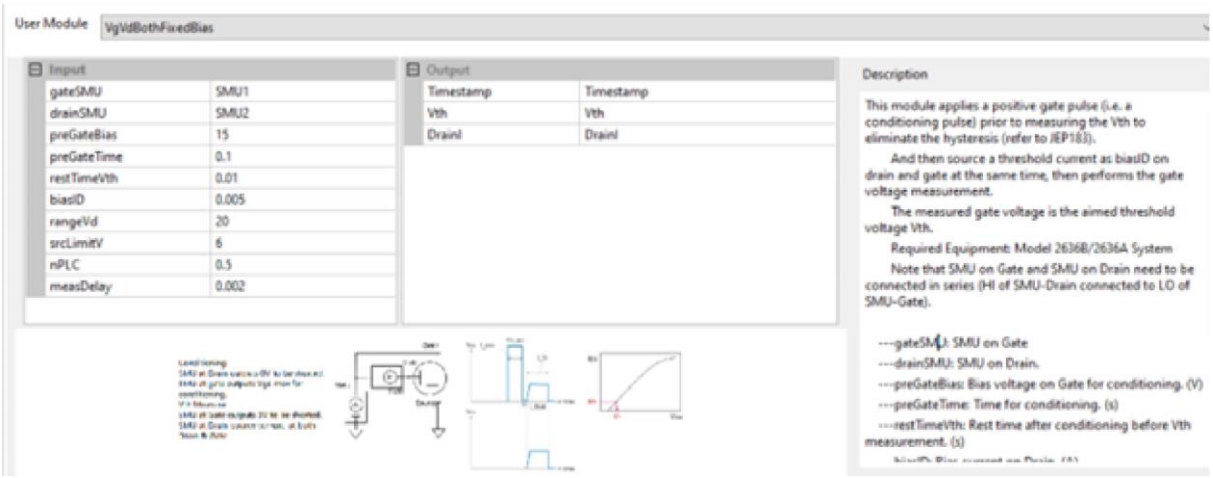
Figure 56: Vg Vd both fixed bias test circuit and timing diagram



Input	Description
gateSMU	SMU on Gate.
drainSMU	SMU on Drain.
preGateBias	Bias voltage on Gate for conditioning.(V)
preGateTime	Time for conditioning.(s)
restTimeVth	Rest time after conditioning before Vth measurement.(s)
biasID	Bias current on Drain. (A)
rangeVd	Voltage range on Drain. (V)
srcLimitI	Source current limit. (V)
nPLC	Number of power line cycle. 0.001~25.
sweepDelay	Idle current level (base level for pulses) (A)

Output	Description
Timestamp	Time stamp reading. (s)
Vth	Threshold voltage. (V)
DrainI	Drain current reading. (A)

Figure 57: Vg Vd both fixed bias GUI



Section 5

ACS Python user library

In this section:

Introduction

ACS has a Python user library (PTMLib) that includes libraries for C-V test, matrix control, scope control, and other external instruments. It is in the following directory:

`\\ACS\library\pyLibrary\PTMLib`.

All test modules in the PTMLib can be imported to a PTM. You can also build a Python library to import and use. For more information, you can refer to Python language Test Module (PTM) configuration in the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01).

Configure a capacitor meter library

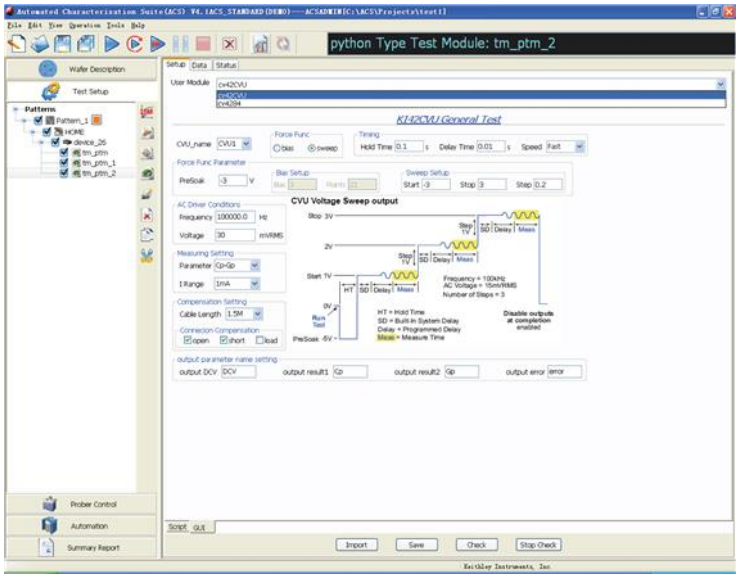
These modules test capacitive parameters at specified frequencies and AC drive voltages, with measurements at the DC voltage bias or sweep.

CV_4200ACVU

To add a CVU to the 4200A-SCS:

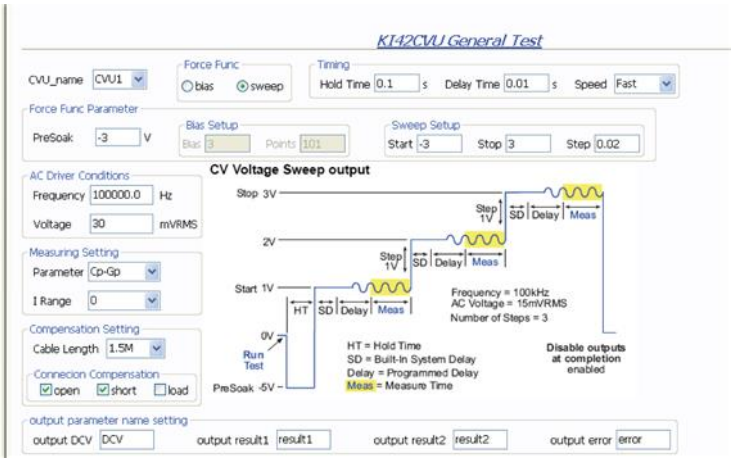
1. Add a PTM to the configuration navigator, then import the `CVITM.py` module from the PTMLib library.
2. From the User Module list, select CVU4200A in the user module to open the Model 4200A-CVU test.

Figure 58: Select CVU4200A user module



The details of the Model 4200A-CVU user interface are indicated in the following figure with a description.

Figure 59: CV_4200ACVU setting example



The user interface inputs are:

CVU_name:	Instrument ID of the Model 4200A-CVU, sub-list CVU1, CVU2, CVU3, CVU4

Force Func: bias or sweep

Timing:

- **Hold Time:** Hold time after force value changed
- **Delay Time:** Delay before each measurement (0 s to 999 s)
- **Speed:** KI_CVU_SPEED_FAST = 0; KI_CVU_SPEED_NORMAL = 1; KI_CVU_SPEED_QUIET = 2

Force Func Parameters:

PreSoak: Force voltage after the test starts and before the measurement sequence.

Bias Setup (enabled by selecting bias in Force Func)

- **Bias:** Force value for the bias
- **Points:** The number of bias points

Sweep Setup (enabled by selecting Sweep in Force Func)

- **Start:** Initial force value for the sweep (0.001 V to 30 V)
- **Stop:** Final force value for the sweep (−30 V to 30 V)
- **Step:** Step force value for the sweep (−30 V to 30 V)

AC Driver Conditions:

- **Frequency:** Frequency of the AC drive. Supported frequency: 10 kHz to 100 kHz in 10 kHz steps, 100 kHz to 1 MHz in 100 kHz steps, 1 MHz to 10 MHz in 1 MHz steps. If an entered value is not a supported frequency, the closest supported frequency is selected (for example, 15 kHz input changes to 20 kHz).
- **Voltage:** Voltage level of the AC drive (10 mV to 100 mV_{RMS}).

Measure Setting:

- **Parameter:** Valid input ['Z,Theta', 'R+jx', 'Cp-Gp', 'Cs-Rs', 'Cp-D', 'Cs-D']

  KI_CVU_TYPE_ZTH = 0

  KI_CVU_TYPE_RJX = 1

  KI_CVU_TYPE_CPGP = 2

  KI_CVU_TYPE_CSRS = 3

  KI_CVU_TYPE_CPD = 4

  KI_CVU_TYPE_CSD = 5
- **I Range:** Current measure range for impedance measurements. Setting the range to zero enables autorange.

Compensation Setting:

- **Cable Length:** Setting for connection compensation. Values from zero to three are valid, but only 0 meters, 1.5 meters, and 3 meters are supported lengths. Any other number from zero to three will be changed to one of the three values. When you do not need compensation, the cable length should be assigned to zero.

Connection Compensation:

- **Open:** Enable or disable compensation constants for an open
- **Short:** Enable or disable compensation constants for a short
- **Load:** Enable or disable compensation constants for a load

Output parameter name setting:

- input the name for the output parameter

Output error list:

Outputs	
0:	OK
-10000	(INVAL_INST_ID): The specified instrument ID does not exist; specified CVU does not exist
-10001	(INVAL_PARAM): An invalid input parameter is specified
-10090:	(GPIB_ERROR_OCCURRED): A GPIB communications error occurred
return dictionary:	
result["DCV"]:	Force DC voltage
result["result1"]:	The first parameter of the result according to the measurement model
result["result2"]:	The second parameter of the result according to the measurement model

Syntax:

```
CVITM.cv42CVU(CVU_name,force_func,preSoak,v_bias,v_biasPts,v_start,v_stop,v_step,hold_time,delay_time,speed,freq_bias,v_AC,meas_param,meas_range,cable_length,isCmpstOpen,isCmpstShort,isCmpstLoad,output_DCV,output_result1,output_result2,output_error)
```

CV_HP4284

To add a CVU HP4284 to the 4200A-SCS:

1. Add a PTM to the configuration navigator, then import the CVITM.py module from the PTMLib library. The CV test user interface will display.
2. Select the CV4284 in the user module to open the Model 4200A-CVU test.

Figure 60: Select HP4284 user module

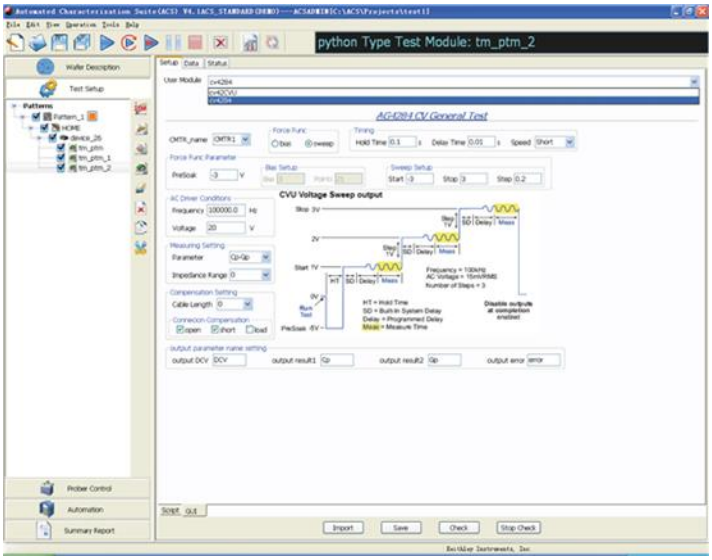
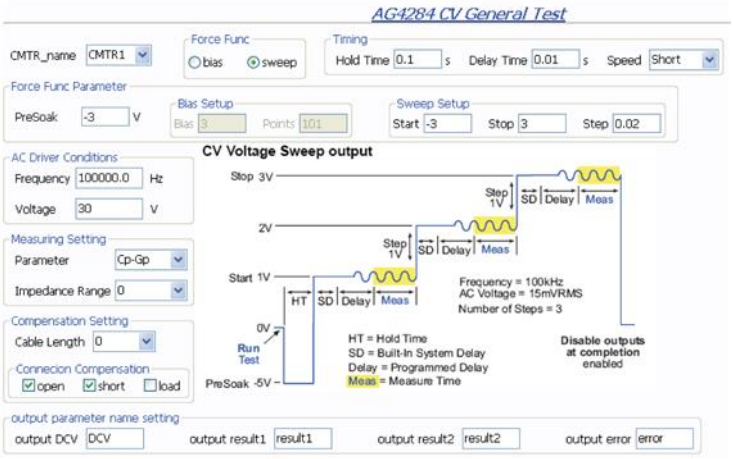


Figure 61: CV_HP4284 setting example



The user interface inputs are:

CMTR_name:	Instrument ID of the CV_HP4284, sub-list CMTR1, CMTR2, CMTR3, CMTR4

Force Func: bias or sweep

Timing:

- Hold Time: Hold time after force value changed
- Delay Time: Delay before each measurement (0 to 999s)
- Speed: KI_CMTR_SPEED_FAST = 0; KI_CMTR_SPEED_NORMAL = 1; KI_CMTR_SPEED_QUIET = 2

Force Func Parameters:

PreSoak: Force voltage after the test starts and before the measurement sequence.

Bias Setup (enabled by selecting bias in Force Func)

- Bias: Force value for the bias
- Points: The number of bias points

Sweep Setup (enabled by selecting Sweep in Force Func)

- Start: Initial force value for the sweep (0.001V to 30V)
- Stop: Final force value for the sweep (-30V to 30V)
- Step: Step force value for the sweep (-30V to 30V)

AC Driver Conditions:

Frequency: Frequency of the AC drive. Supported frequency: 10 kHz to 100 kHz in 10 kHz steps, 100 kHz to 1 MHz in 100 kHz steps, 1 MHz to 10 MHz in 1 MHz steps. If an entered value is not a supported frequency, the closest supported frequency will be selected (for example, 15 kHz input will change to 20 kHz).

- Voltage: Voltage level of the AC drive (10 mV to 100 mVRMS).

Measure Setting:

- Parameter: Valid input ['Z,Theta', 'R+jx', 'Cp-Gp', 'Cs-Rs', 'Cp-D', 'Cs-D']

  KI_AGCV_TYPE_ZTR = 0 "ZTR"

  KI_AGCV_TYPE_RX = 1 "RX"

  KI_AGCV_TYPE_CPG = 2 "CPG"

  KI_AGCV_TYPE_CSRS = 3 "CSRS"

  KI_AGCV_TYPE_CPD = 4 "CPD"

  KI_AGCV_TYPE_CSD = 5 "CSD"
- Impedance Range: Current measure range for impedance measurements. Valid values for this parameter are 0 (Auto), 100, 300, 1000, 3000, 10000, 30000, and 100000 Ohms.

Compensation Setting:

- Cable Length: Setting for connection compensation. Values from zero to three are valid, but only 0 meters, 1.5 meters, and 3 meters are supported lengths. Any other number from zero to three will be changed to one of the three values. When you do not need compensation, the cable length should be assigned to zero.

Connection Compensation:

- Open: Enable or disable compensation constants for an open
- Short: Enable or disable compensation constants for a short
- Load: Enable or disable compensation constants for a load

Output parameter name setting:

- input the name for the output parameter

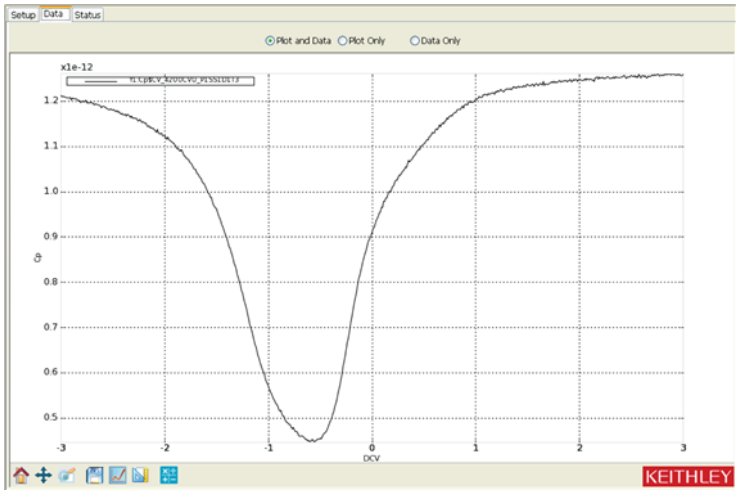
Output error list:

Outputs	
`0:`	OK
`-10000`	`(INVAL_INST_ID):` The specified instrument ID does not exist; specified CVU does not exist
`-10001`	`(INVAL_PARAM):` An invalid input parameter is specified
`-10090:`	`(GPIB_ERROR_OCCURRED):` A GPIB communications error occurred
return dictionary:	
`result["DCV"]:`	Force DC voltage
`result["result1"]:`	The first parameter of the result according to the measurement model
`result["result2"]:`	The second parameter of the result according to the measurement model

Syntax:

```
CVITM.cv4284
(CMTR_name,force_func,preSoak,v_bias,v_biasPts,v_start,v_stop,v_step
,hold_time,delay_time,speed,freq_bias,v_AC,meas_param,meas_range,cab
le_length,isCmpstOpen,isCmpstShort,isCmpstLoad,output_DCV,output_res
ult1,output_result2,output_error)
```

Figure 62: CV_HP4284 Data tab



Configure a switch matrix library

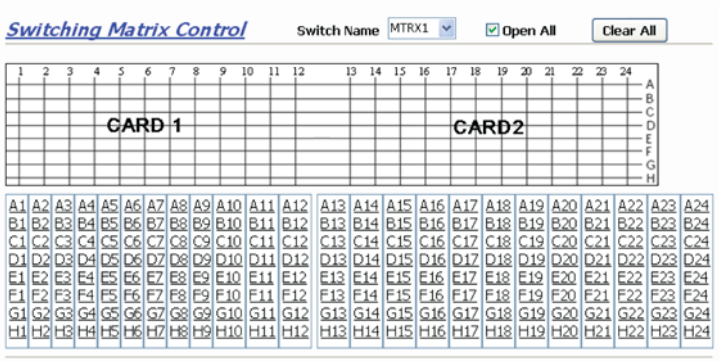
To configure a switch matrix library, you will need to add a PTM to the configuration navigator. See the following two topics, Switch_Control (on page 5-8), and KISeries 3700 System Switch (on page 5-10), for more specific information.

Switch_Control

To use the switch control:

1. Add a PTM to the configuration navigator.
2. Import the `swichctrl.py` module from the PTMLib library.

Figure 63: Switch_Control GUI



This module connects matrix row terminals and column pins, based on the row list and column list. It supports two cards in one switch controller.

Instrument: Keithley Instruments Switch Matrix 707A, 708A

NOTE

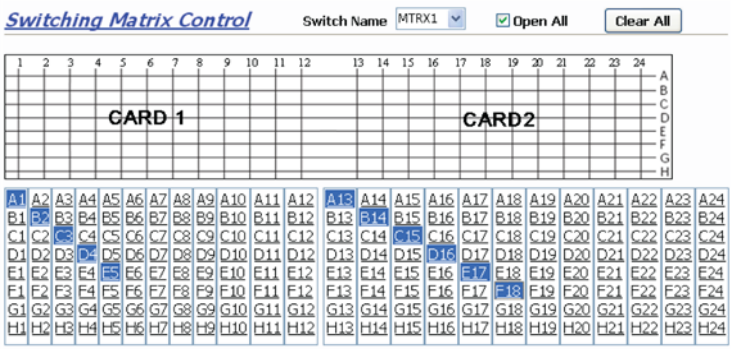
The Model 708A module can control two cards. The cards do not need to be configured in the hardware configuration panel.

Inputs	
`switch_name (int):`	This is the global name that is displayed in the hardware configuration panel
`open_all (int):`	A flag that controls if the switch matrix is first cleared before making any new connections 1, all previous connections are cleared 0, they are left intact
`rowlist (list):`	Matrix row name that will be closed.['A','B']
`collist (list):`	Matrix column name that will be closed.['1','2']

In the user interface, you can control the matrix:

- Select the switch matrix in the list for the Switch Name.
- Select the cells on the panel and the related rows and columns of the matrix will connect. For example, select A1, and the 1 column and A row will connect. The corresponding cell will highlight. Select the highlighted cells again, and the connections will be canceled.
- The Clear All function will clear all connections.
- If the Open All option is selected, the matrix will open all old connections before connecting.

Figure 64: Switch_Control setting example



KISeries 3700 System Switch

To use a Keithley 3700A System Switch:

1. Add a PTM to the configuration navigator.
2. Import the `MDD.py` module from the PTMLib library.

Instrument: Keithley Instruments Series 3700A System Switch/Multimeter Cards

This module supports two types of cards: 6×16, High Density, Matrix Card (Model 3730) and Dual 1×30 Multiplexer Card (3720).

Figure 65: Series 3700 System GUI

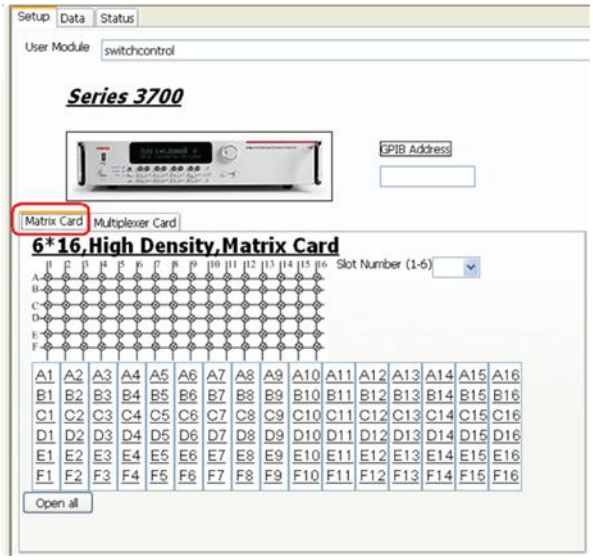
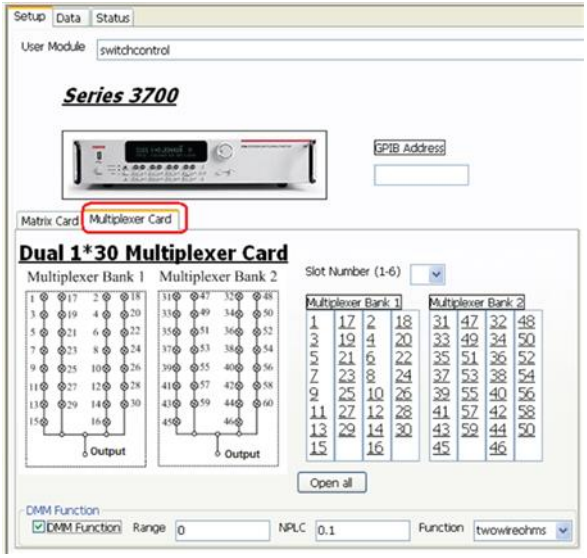


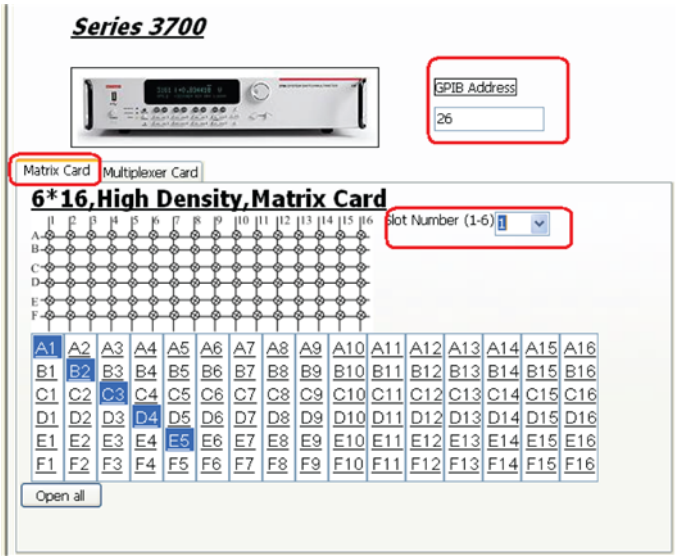
Figure 66: Series 3700 System Multiplexer GUI



To control the Model 3700A matrix from the user interface:

1. Input the GPIB address number in the GPIB edit box.
2. Select the Matrix Card tab.
3. Select the slot number from 1 to 6.
4. Select the cells on the panel and the related rows and columns of the matrix will connect. For example, select A1, and the 1 column and the A row will connect. The corresponding cell will highlight. Select the highlighted cells again, and the connections will be cancelled.
5. If you want to clear all connections, select **Open all**.

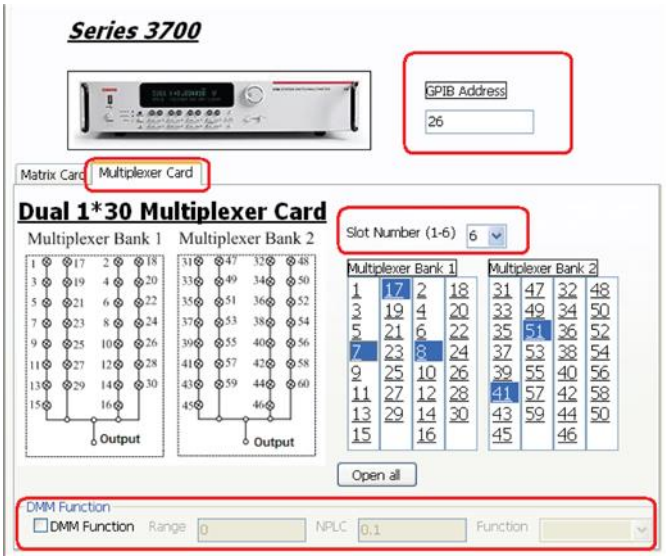
Figure 67: Matrix control setting example



To control the multiplexer card from the user interface:

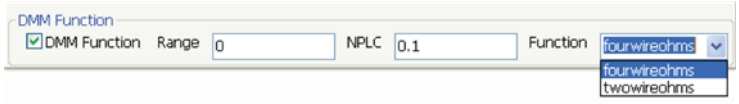
1. Input the GPIB address number in the GPIB edit box.
2. Select the **Multiplexer Card** tab.
3. Select the **Slot Number** from 1 to 6.
4. Select the cells on the panel and the related rows and columns of the matrix will connect. For example, select the A1, and the 1 column and the A row will connect. The corresponding cell will highlight. Select the highlighted cells again, and the connections will be cancelled.
5. If you want to clear all connections, select **Open all**.

Figure 68: Multiplexer control setting example



If want to set up a DMM function test, select the **DMM Function**, then set the range (0 means autorange), NPLC, and select the function from the menu (**fourwireohms** or **twowireohms**).

Figure 69: DMM setting example



Script **inputs**	
`GPIB_Address:`	GPIB address
`Open_all:`	Open all the channels
`S1Channel1:`	Channel list for 6*16 High Density, Matrix Card
`S1Channel1`	
`S1Channel2`	
`S1Channel3`	
`S1Channel4`	
`......`	
`S1Channel16`	
`List_1:`	Channel list for Multiplexer Card
`List_1`	
`List_2`	
`......`	
`List_8`	
`SlotNumberCard1:`	Slot number for Matrix Card
`SlotNumberCard2:`	Slot number for Multiplexer Card
`ModuleCardNum:`	

Configure a scope library

To add and configure an oscilloscope library, you need to add a PTM to the configuration navigator. See TEKSCOPE_ReadWave (on page 5-13) for more information.

TEKSCOPE_ReadWave

To add a Tektronix oscilloscope:

1. Add a PTM to the configuration navigator.
2. Import the `TEKSCOPE.py` module from the PTMLib library.

This module reads data from one channel at a time. Some modifications are needed to enable it to read data from more channels simultaneously.

Instrument: TEKSCOPE

Figure 70: TEKSCOPE read wave test module GUI

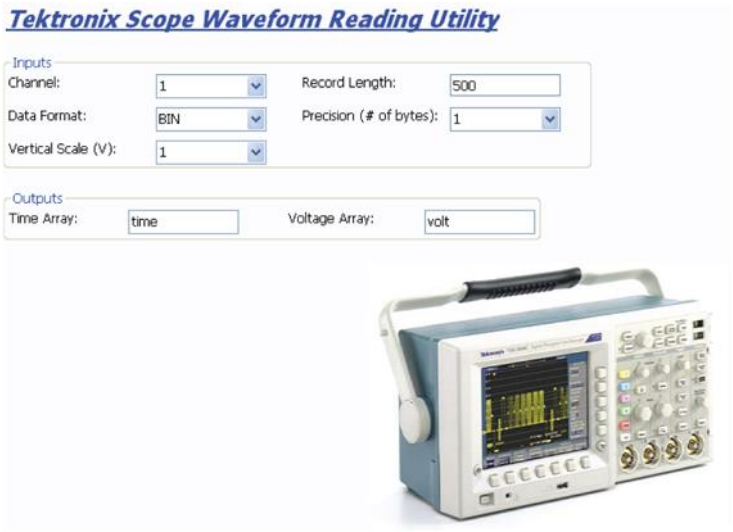
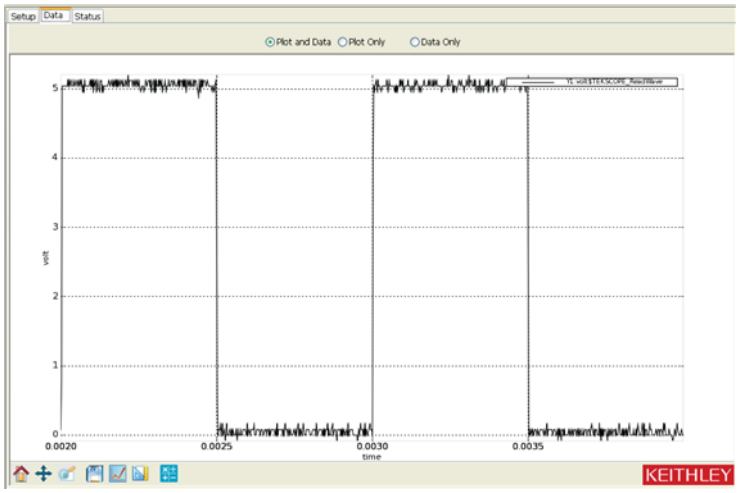


Figure 71: Waveform reading data



Configure a Series 23x library

The following modules will help you to configure your Keithley Instruments Model 236, Model 237, or Model 238 library to bias voltage, take current readings, sweep current, sweep voltage, and take high-voltage measurements for the drain current while forcing drain voltage and stepping the gate voltage. See the following topics BiasVolt_SampleCurr (on page 5-15), Sweepsystem computer_23x (on page 5-16), SweepVolt_23x (on page 5-18), VdsIs_237 (on page 5-20) for more information.

BiasVolt_SampleCurr

This module is used to bias voltage and take current readings for Model 236, Model 237, or Model 238.

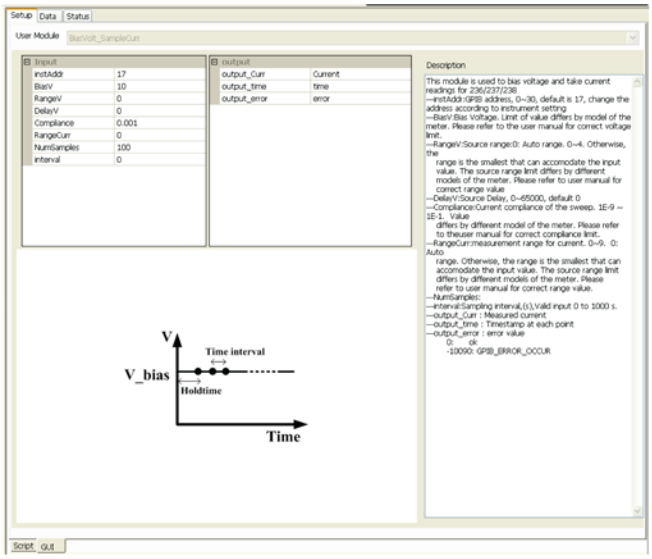
Instrument: Keithley Instruments Model 236, Model 237, or Model 238.

Inputs	
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`BiasV:`	Bias Voltage
`RangeV:`	Voltage range. If zero is selected, the instrument will autorange
`DelayV:`	Voltage delay. Zero through 65000 is the time in seconds for a delay and the default is zero (0)
`Compliance:`	Current-sweep compliance (1E-9 through 1E-1)
`RangeCurr:`	Current measurement range (zero through nine is the range). If zero is selected, the instrument will autorange

NumSamples	
`interval:`	Sampling interval. Valid input is zero to 1000 s

Outputs	
`output_Curr:`	Measured Current output
`output_time:`	Timestamp at each point
`output_error:`	Error value
`0:`	OK
`-10090:`	GPIB_ERROR_OCCUR

Figure 72: 23x Bias V Sample standard GUI



Sweepsystem computer_23x

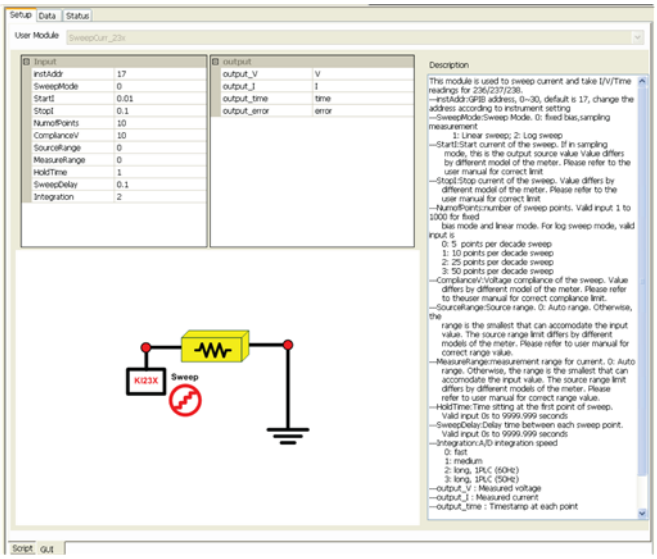
This module is used to sweep current and take current, voltage, and time readings for the Model 236, Model 237, or Model 238.

Instrument: Keithley Instruments Model 236, Model 237, or Model 238 source measure units.

Inputs	
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`SweepMode:`	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep
`StartI:`	Start the sweep current. If in sampling mode, this is the output source value
`StopI:`	Stop the sweep current
`NumofPoints:`	Number of sweep points. Valid input is 1 to 1000 for fixed bias mode and linear mode. For the log sweep mode, valid input is: 0: 5 points per decade sweep 1: 10 points per decade sweep 2: 25 points per decade sweep 3: 50 points per decade sweep
`ComplianceV:`	Voltage-sweep compliance
`sourceRange:`	Source range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value.
`MeasureRange:`	Measurement range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value.
`HoldTime:`	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 s.
`SweepDelay:`	Delay time between each sweep point. Valid inputs are zero to 9999.999 s.
`Integration:`	Analog/digital integration speed: 0: fast 1: medium 2: long, 1PLC (60 Hz) 3: long, 1PLC (50 Hz)

Outputs	
`output_V:`	Measured voltage
`output_I:`	Measured current
`output_time:`	Timestamp at each point
`output_error:`	Error value 0: OK -10090: GPIB_ERROR_OCCU -10100: INVAL_PARAM

Figure 73: 23x Sweep standard GUI



SweepVolt_23x

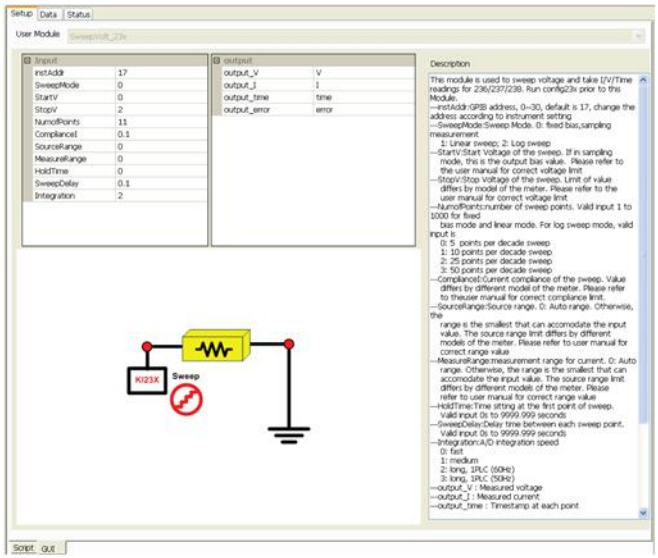
This module is used to sweep voltage and take current, voltage, and time readings for the Model 236, Model 237, or Model 238.

Instrument: Keithley Instruments Model 236, Model 237, or Model 238.

Inputs	
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`SweepMode:`	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep
`StartV:`	Start the sweep voltage. If in sampling mode, this is the output bias value
`StopV:`	Stop the sweep voltage
`NumofPoints:`	Number of sweep points. Valid input is 1 to 1000 for fixed bias mode and linear mode. For the log sweep mode, valid input is: 0: 5 points per decade sweep 1: 10 points per decade sweep 2: 25 points per decade sweep 3: 50 points per decade sweep
`ComplianceI:`	Current-sweep compliance
`SourceRange:`	Source range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`MeasureRange:`	Measurement range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`HoldTime:`	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 s
`SweepDelay:`	Delay time between each sweep point. Valid inputs are zero to 9999.999 s
`Integration:`	Analog/digital integration speed: 0: fast 1: medium 2: long, 1PLC (60 Hz) 3: long, 1PLC (50 Hz)

Outputs	
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`output_V:`	Measured voltage
`output_I:`	Measured current
`output_time:`	Timestamp at each point
`output_error:`	Error value
`0`	OK
`-1`	23x not found on GPIB
`-10000`	(INVAL_INST_ID) The specified instrument ID does not exist
`-10090`	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
`-10091`	(GPIB_TIMEOUT) A timeout occurred during communications
`-10100`	(Invalid Parameter) An error occurred on an input parameter

Figure 74: 23x SweepV standard GUI



VdsIs_237

This module is used to take high-voltage measurements of the drain current while forcing drain voltage and stepping the gate voltage.

Instruments: Keithley Instruments Model 236, Model 237, or Model 238, and Model 4200A Semiconductor Characterization System.

Inputs	
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`instAddr:`	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting
`GateSMU:`	The system terminal connected to the MOSFET gate. If 'GNDU' is chosen, the terminal should be connected to ground manually
`sourceSMU:`	The system terminal connected to the MOSFET source. If 'GNDU' is chosen, the terminal should be connected to ground manually
`SubSMU:`	The system terminal connected to the MOSFET substrate. If 'GNDU' is chosen, the terminal should be connected to ground manually
`WellSMU:`	The system terminal connected to the MOSFET well. If 'GNDU' is chosen, the terminal should be connected to ground manually. If there is not a well terminal, choose 'NONE'
`VgStart:`	Start the gate voltages
`VgStop:`	End the gate voltages
`VgPoint:`	Number of forced intervals
`VdStart:`	Start the drain voltages
`VdStop:`	End the drain voltages
`VdPoint:`	Number of forced drain voltage intervals
`IdLimit:`	Current limit on sites (measured in amps)
`Integration:`	Analog/digital integration speed 0: fast 1: medium 2 long, 1PLC (60 Hz) 3: long, 1PLC (50 Hz)
`DelayTime:`	Delay time of measurements (seconds)
`VscForce:`	Source voltage bias force
`VsbForce:`	Substrate voltage bias force
`VwForce:`	Voltage bias force to Well
`VgMsrFlag:`	Flag to determine if the gate voltage is measured
`IgMsrFlag:`	Flag to determine if the gate current is measured
`VscMsrFlag:`	Flag to determine if the source voltage is measured
`IscMsrFlag:`	Flag to determine if the source current is measured

Inputs	
`VsbMsrFlag:`	Flag to determine if the substrate voltage is measured
`IsbMsrFlag:`	Flag to determine if the substrate current is measured
`VwMsrFlag:`	Flag to determine if the well voltage is measured
`IwMsrFlag:`	Flag to determine if the well current is measured

Out**puts**	
`0`	OK
`-1`	23x not found on GPIB
`-10000`	(INVAL_INST_ID) The specified instrument ID does not exist
`-10090`	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
`-10091`	(GPIB_TIMEOUT) A timeout occurred during communications
`-10100`	(Invalid Parameter) An error occurred on an input parameter

Figure 75: Model 237 VdsId standard GUI

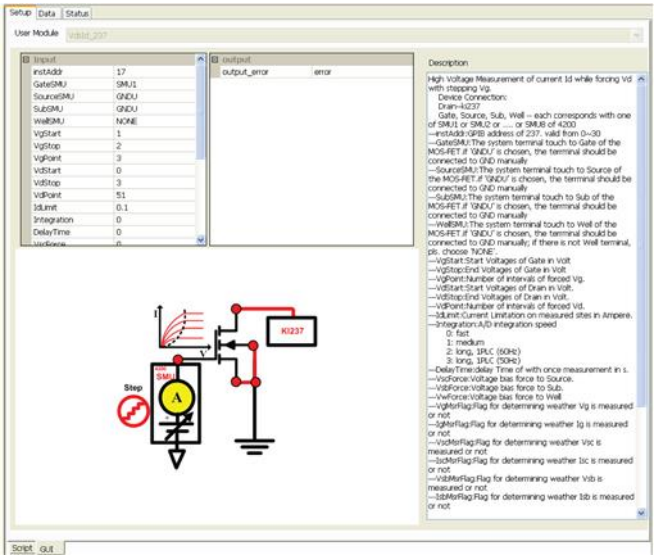
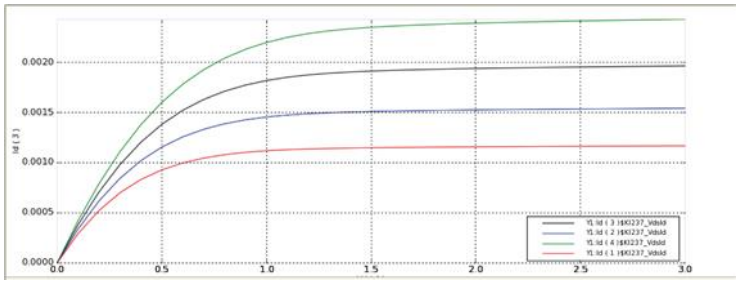


Figure 76: Model 237 VdsId test result



Configure a Series 3700 system switch DMM library

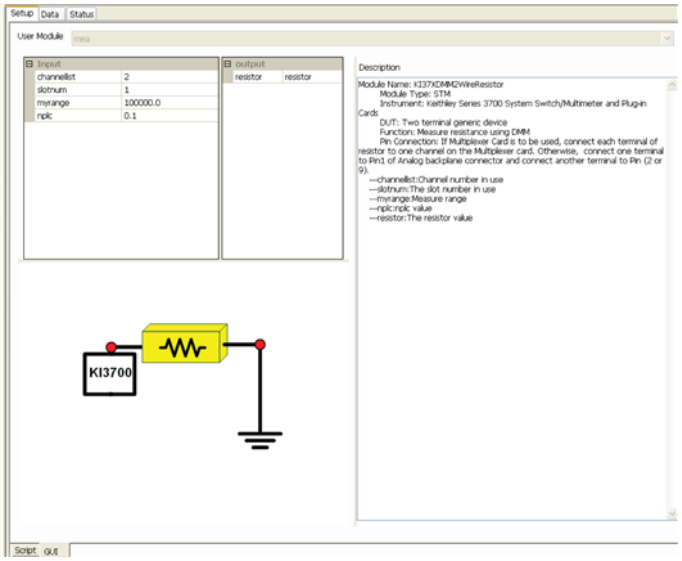
To configure a Series 3700 system switch DMM library, you will need to add a PTM to the configuration navigator. See the following two topics, Series 3700 System Switch DMM two-wire (on page 5-22) and Series 3700 System Switch DMM four-wire (on page 5-23), for more specific information.

Series 3700 System Switch DMM two-wire

To add a Series 3700 System Switch:

1. Add a PTM to the configuration navigator.
2. Import the `Gpibresistor.py` module from the PTMLib library.

Figure 77: Series 3700 Switch DMM 2-wire standard GUI



Instrument: Keithley Instruments Series 3700 System Switch/Multimeter and plug-in cards.

DUT: Two-terminal generic device.

Function: Measure resistance using a DMM.

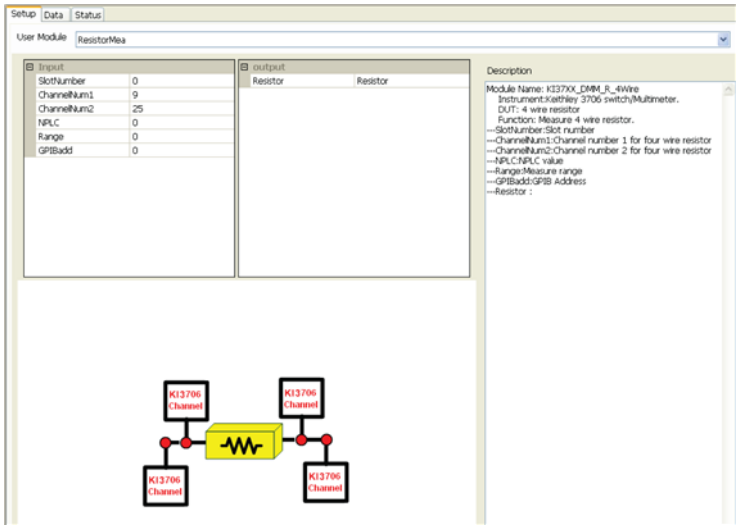
Pin connections: If the multiplexer card is used, connect each terminal of the resistor to one channel on the multiplexer card. Otherwise, connect one terminal to Pin1 of the analog backplane connector and connect another terminal to Pin (2 or 9).

Series 3700 System Switch DMM four-wire

To add a 3700 System Switch:

1. Add a PTM to the configuration navigator.
2. Import the `FourWireResistor.py` module from the PTMLib library.

Figure 78: Series 3700 Switch DMM 4-wire standard GUI



Instrument: Keithley Instruments Series 3700 System Switch/Multimeter and plug-in cards.

DUT: Four-terminal generic device.

Function: Measure resistance using a DMM.

Pin connections: If a multiplexer card is used, a channel pair is used for 4-wire measurement; channels 1 through 20 are used as the INPUT terminals and channels 21 through 40 are used as the SENSE terminals. Otherwise, connect the Input HI terminal of the resistor to Pin1 of the analog backplane connector, the Input LO terminal to Pin (2 or 9), the Sense HI to Pin3, and Sense LO to Pin4.

Configure a Series 2400 SourceMeter instruments library

To configure a Series 2400 SourceMeter instruments library, add a PTM to the configuration navigator. See the following topics for help to choose the appropriate test module from the test module list:

Series 2400 drain-current test

To add a Series 2400:

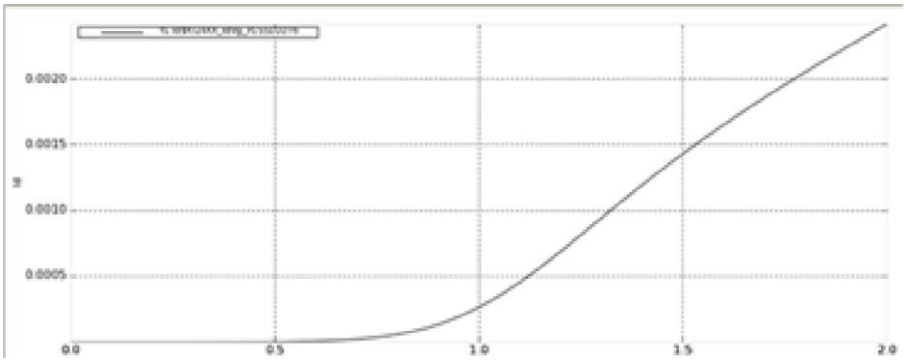
1. Add a PTM to the configuration navigator.
2. Import the `HiPower_24.py` module from the PTMLib library. Choose the appropriate test module from test module list.

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to test the drain-current at a specified drain-voltage value during a gate-voltage sweep.

Pin connections: Sweep the gate and bias the drain. The bulk and source are connected to ground if there is no applied voltage.

Figure 79: Series 2400 instrument IdVg test results



Results

- Get the drain-current measurement during the gate-voltage sweep.
- Get the Vtx and Vt0 results.

Inputs

`drain_addr (int):`	Model 2430 GPIB drain-terminal address
`gate_addr (int):`	Series 2400 SourceMeter GPIB gate-terminal address
`vg_start (double):`	Start the gate-voltage pulse (valid input is from -200V to 200V)
`vg_stop (double):`	Stop the gate-voltage pulse (valid input is from -200V to 200V)
`points (int):`	Number of drain-sweep points (valid input 1 to 2500)
`vd (double):`	Drain the bias voltage
`hold_time (double):`	Hold time in second before gate sweep. Valid inputs are zero to 9999.999 s
`delay_time (double):`	Delay time between each sweep point. Valid inputs are zero to 9999.999 s
`limiti (double):`	Drain-voltage force compliance value (valid input is from -10A to 10A)
`rangei (double):`	Range for drain-current measurement. For pulse mode, autorange is not allowed (valid input is from -10 through 10)
`plc (double):`	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10)

Outputs

`Id`	(D_ARRAY_T) Drain-current measured at gate sweep voltage
`Vg`	(D_ARRAY_T) Gate-voltage programmed
`Gm`	(D_ARRAY_T) Gm=dId/dVg
`Vtx`	(double*) Vtx= Vt0-Vs/2

Error value	Description
0	OK
-1	Series 2400 SourceMeter instruments not found on GPIB
-200	Instrument initialize error
-10000	(INVAL_INST_ID) The specified instrument ID does not exist
-10100	(INVAL_PARAM) Parameter setting error occurred
-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 80: Series 2400 instruments IdVg

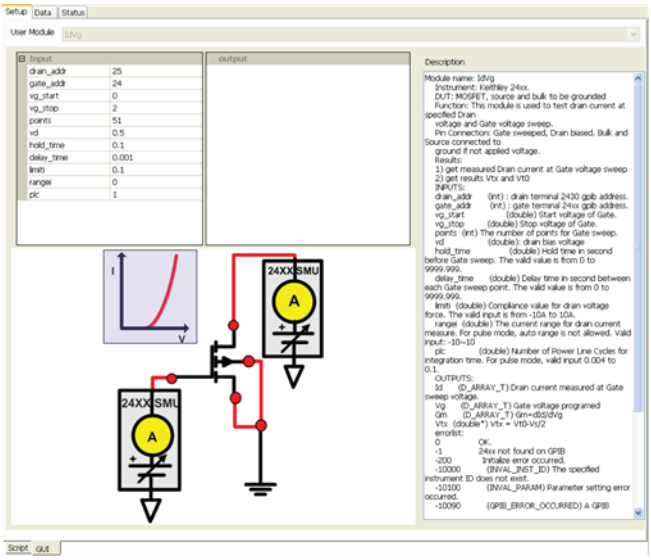
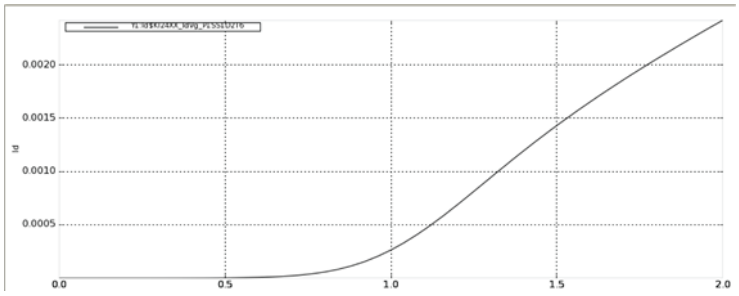


Figure 81: Series 2400 instruments IdVg test results



Model 2430 gate-voltage sweep test

Module type: PTM

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to test the drain-current during a gate-voltage sweep, at a specified drain-voltage, while using the Model 2430 SourceMeter (controlled over a GPIB bus only), with a measurement at the drain-terminal in sweep pulse mode.

Pin connections: Sweep the gate and bias the drain. The bulk and source are connected to ground if there is no applied voltage.

Results

- Get the drain-current measurement during the gate-voltage sweep, with the drain in pulse mode.
- Get Vtx and Vt0 results.

Inputs

NOTE

Pulse width should be longer than 200 us if the measurement is in pulse mode. If the pulse width is shorter than the measurement time (which is based on NPLC and line frequency) the pulse width will broaden automatically.

`drain_addr` (int):	Model 2430 GPIB drain-terminal address
`gate_addr` (int):	Series 2400 SourceMeter GPIB gate-terminal address
`vg_start` (double):	Start the gate-voltage pulse (valid input is from -200V to 200V)
`vg_stop` (double):	Stop the gate-voltage pulse (valid input is from -200V to 200V)
`points` (int):	Number of drain-sweep points (valid input 1 to 2500)
`vd` (double):	Drain the bias voltage
`hold_time` (double):	Hold time in second before gate sweep. Valid inputs are zero to 9999.999 s
`delay_time` (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 s
`limiti` (double):	Drain-voltage force compliance value (valid input is from -10A to 10A)
`rangei` (double):	Range for drain-current measurement. For pulse mode, autorange is not allowed (valid input is from -10 through 10)
`plc` (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10)

`pulse_width` (double):	Duration of the output ON time (valid value is from 0.15 ms to 5 ms)
`pulse_delay` (double):	Duration of the output OFF time (valid value is from zero to 9999.999 s)

Outputs

`Id`	(D_ARRAY_T) Drain-current measured at gate sweep voltage
`Vg`	(D_ARRAY_T) Gate-voltage programmed
`Gm`	(D_ARRAY_T) Gm=dId/dVg
`Vtx`	(double*) Vtx= Vt0-Vs/2
`Vt0`	(double*) Calculate Gm=dId/dVg. Find Gmmax and extrapolate back to Ids=0 to find Vt0

`Error:`	Error value	
	0	OK
	-1	Series 2400 SourceMeter instruments not found on GPIB
	-2	2430 not found on GPIB
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 82: Model 2430 IdVg Pulse

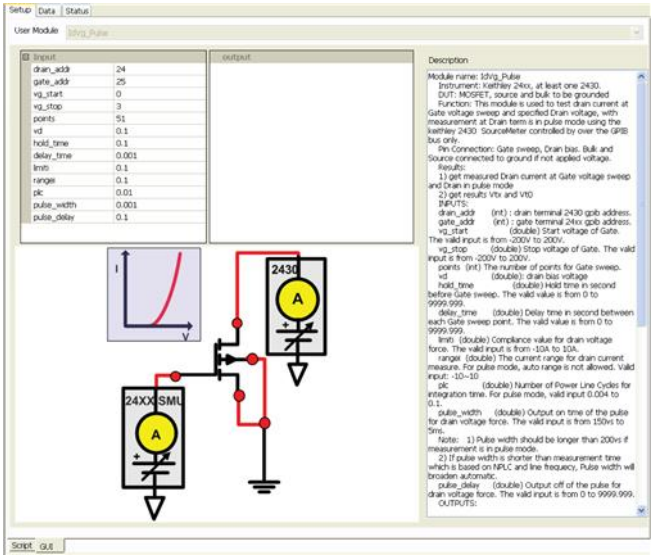
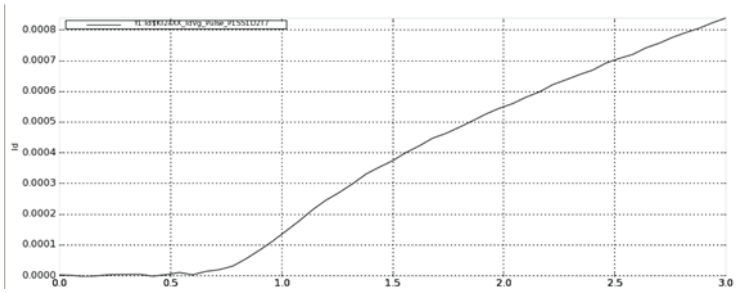


Figure 83: Series 2400 instruments IdVg Pulse test results



Series 2400 gate-voltage test

Module type: PTM

Instrument: Keithley Instruments Model 2430 SourceMeter.

DUT: MOSFET, source and bulk are grounded.

Function: This module is used to test the drain-current at a specified gate-voltage during a drain-voltage sweep.

Pin connections: Sweep the drain and bias the gate. The bulk and source are connected to ground if there is no applied voltage.

Results: Get the drain-current measurement during the drain-voltage sweep at 10 gate-bias voltages.

Inputs

`drain_addr` (int):	Model 2430 GPIB drain-terminal address
`gate_addr` (int):	Series 2400 SourceMeter GPIB gate-terminal address
`vd_start` (double):	Start the voltage-drain pulse
`vd_stop` (double):	Stop the voltage-drain
`points` (int):	Number of drain-sweep points (valid input 1 to 2500)
`limiti` (double):	Drain-voltage force compliance value (valid input is from -10A to 10A)
`rangei` (double):	Range for drain-current measurement. For pulse mode, autorange is not allowed (valid input is from -10 through 10)
`plc` (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.004 to 0.10)
`vg_start` (double):	Start the voltage-gate pulse
`vg_stop` (double):	Stop the voltage-gate pulse
`vg_step` (double):	Step the voltage-gate pulse
`hold_time` (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 s
`delay_time` (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 s

Outputs

`Vd`	(D_ARRAY_T) Drain-voltage programmed
`Id1`	(D_ARRAY_T) Drain-current measured at the first gate bias voltage
`Id2`	(D_ARRAY_T) Drain-current measured at the second gate bias voltage
`Id3`	(D_ARRAY_T) Drain-current measured at the third gate bias voltage
`Id4`	(D_ARRAY_T) Drain-current measured at the fourth gate bias voltage

`Error:`	Error value	
	0	OK
	-1	Series 2400 SourceMeter instruments not found on GPIB
	-2	2430 not found on GPIB
	-200	Instrument initialize error
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 84: Series 2400 instruments IdVd

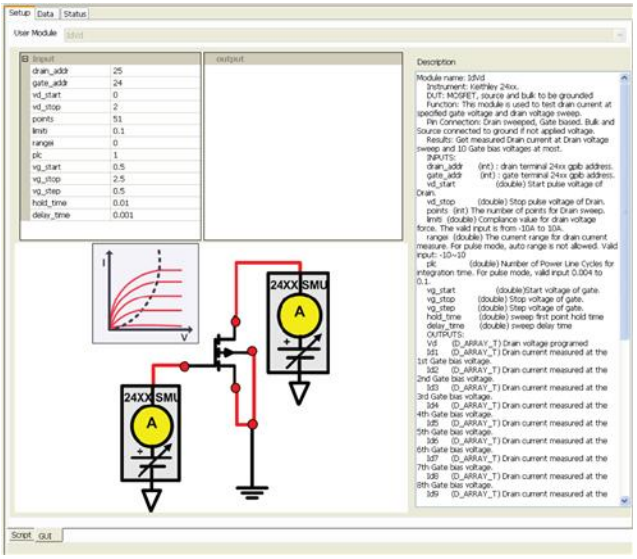
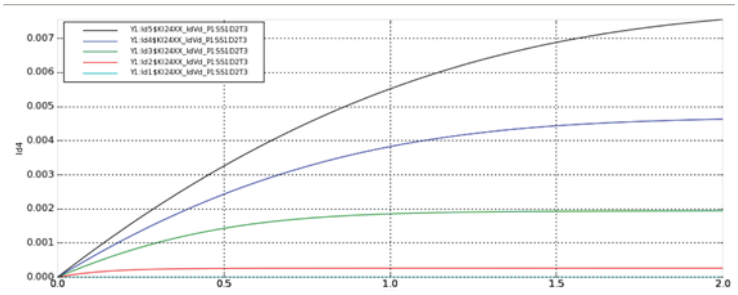


Figure 85: Series 2400 instruments IdVd test results



Model 2430 drain-voltage sweep test

Module Type: PTM

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to perform a voltage pulse and current measurement with a Model 2430 in pulse mode (controlled over a GPIB bus only).

Function: This module is used to test the drain-current at a specified gate-voltage, during a drain-voltage sweep, while using the Model 2430 SourceMeter (controlled over a GPIB bus only), with a measurement at the drain terminal in sweep pulse mode.

Inputs

`drain_addr` (int):	Model 2430 GPIB drain-terminal address
`gate_addr` (int):	Series 2400 SourceMeter GPIB gate-terminal address
`vd_start` (double):	Start the voltage-drain pulse
`vd_stop` (double):	Stop the voltage-drain
`points` (int):	Number of drain-sweep points (valid input 1 to 2500)
`limiti` (double):	Drain-voltage force compliance value (valid input is from -10A to 10A)
`rangei` (double):	Range for drain-current measurement. For pulse mode, autorange is not allowed (valid input is from -10 through 10)
`plc` (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.004 to 0.10)
`vg_start` (double):	Start the voltage-gate pulse
`vg_stop` (double):	Stop the voltage-gate pulse
`vg_step` (double):	Step the voltage-gate pulse
`pulse_width` (double):	Duration of the output ON time (valid value is from 0.15 ms to 5ms)
`pulse_delay` (double):	Duration of the output OFF time (valid value is from zero to 9999.999 s)

Outputs

`Vd`	(D_ARRAY_T) Drain-voltage programmed
`Id1`	(D_ARRAY_T) Drain-current measured at the first gate bias voltage

`Error:`	Error value	
	0	OK
	-1	Series 2400 SourceMeter instruments not found on GPIB
	-2	2430 not found on GPIB
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 86: Series 2400 instruments IdVd pulse

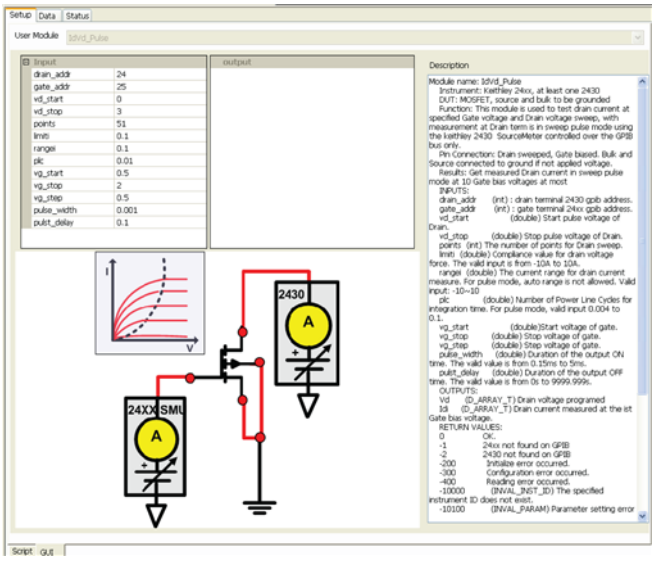
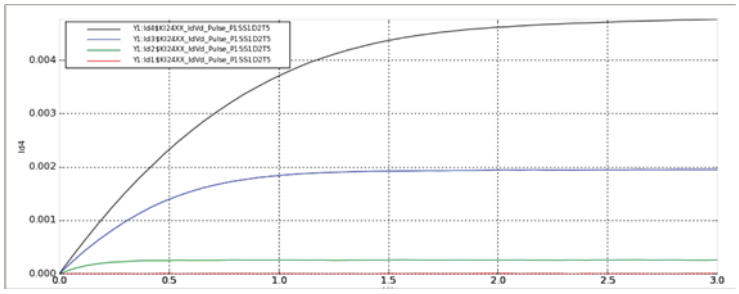


Figure 87: Series 2400 instruments IdVd pulse test results



Model 2430 voltage-pulse test

Module Type: PTM

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to perform a voltage pulse and current measurement with a Model 2430 in pulse mode (controlled over a GPIB bus only).

Results: Force the voltage pulse.

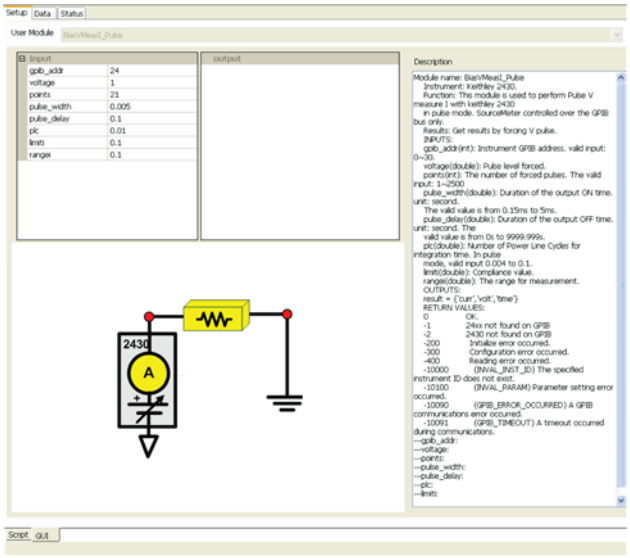
Inputs

`gpib_addr` (int):	Instrument GPIB address. Valid input: 1 through 30
`voltage` (double):	Pulse level forced
`points` (int):	Number of forced pulses (valid input 1 to 2500)
`pulse_width` (double):	Duration of the output ON time (valid value is from 0.15 ms to 5 ms)
`pulse_delay` (double):	Duration of the output OFF time (valid value is from zero to 9999.999 s)
`plc` (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10)
`limiti` (double):	Current-sweep compliance value
`rangev` (double):	Range for voltage measurement. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value

Outputs

`Error:`	Error value	
	0	OK
	-1	Series 2400 SourceMeter instruments not found on GPIB
	-2	2430 not found on GPIB
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications
`time`		
`I`		
`V`		

Figure 88: Series 2400 instruments BiasV pulse standard GUI



Model 2430 current-pulse test

Module name: BiasIMeasV_Pulse

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to perform a current pulse and voltage measurements with a Model 2430 in pulse mode (controlled over a GPIB bus only).

Results: Force the current pulse.

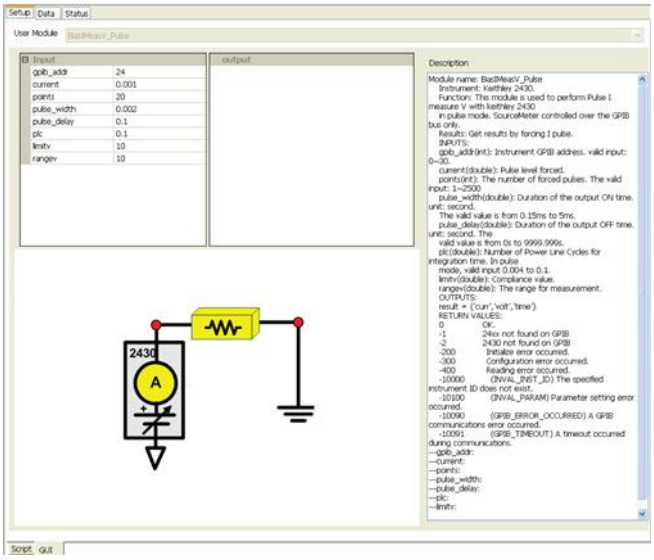
Inputs

`gpib_addr` (int):	Instrument GPIB address. Valid input: 1 through 30
`current` (double):	Pulse level forced
`points` (int):	Number of forced pulses (valid input 1 to 2500)
`pulse_width` (double):	Duration of the output ON time (valid value is from 0.15 ms to 5 ms)
`pulse_delay` (double):	Duration of the output OFF time (valid value is from zero to 9999.999 s)
`plc` (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10)
`limiti` (double):	Current-sweep compliance value
`rangev` (double):	Range for voltage measurement. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value

Outputs

I		
V		
time		
Error:	Error value	
	0	OK
	-1	Series 2400 SourceMeter instruments not found on GPIB
	-2	2430 not found on GPIB
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 89: Series 2400 instruments BiasI pulse standard GUI



Series 2400 voltage-sweep test

Module name: SweepV_MeasI

Instrument: Keithley Instruments Models 2400/2410/2420/2425/2430 SourceMeter.

Function: This module is used to sweep the voltage signal and take current, voltage and time readings for the Models 2400/2410/2420/2425/2430 instruments.

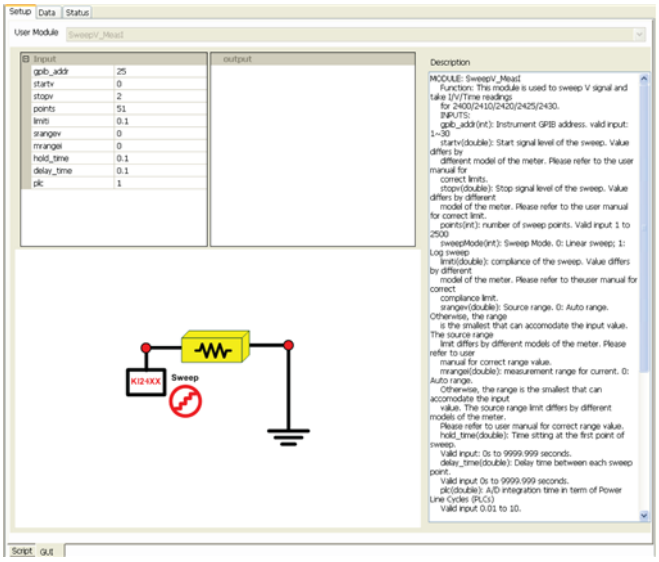
Inputs

`gpib_addr` (int):	Instrument GPIB address. Valid input: 1 through 30
`startv` (double):	Start the voltage sweep signal
`stopv` (double):	Stop the voltage sweep signal
`points` (int):	Number of sweep points (valid input 1 to 2500)
`sweepMode` (int):	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep.
`limiti` (double):	Current-sweep compliance value.
`srangev` (double):	Source range for voltage current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`mrangei` (double):	Measurement range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`hold_time` (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 s
`delay_time` (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 s
`plc` (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.01 to 10)

Outputs

I		
V		
time		
Error:	Error value	
	0	OK
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 90: Series 2400 instruments SweepV standard GUI



Series 2400 current-sweep test

Module name: SweepI_MeasV

Instrument: Keithley Instruments Models 2400/2410/2420/2425/2430 SourceMeter.

Function: This module is used to sweep the current signal and take current, voltage, and time readings for the Models 2400/2410/2420/2425/2430 instruments.

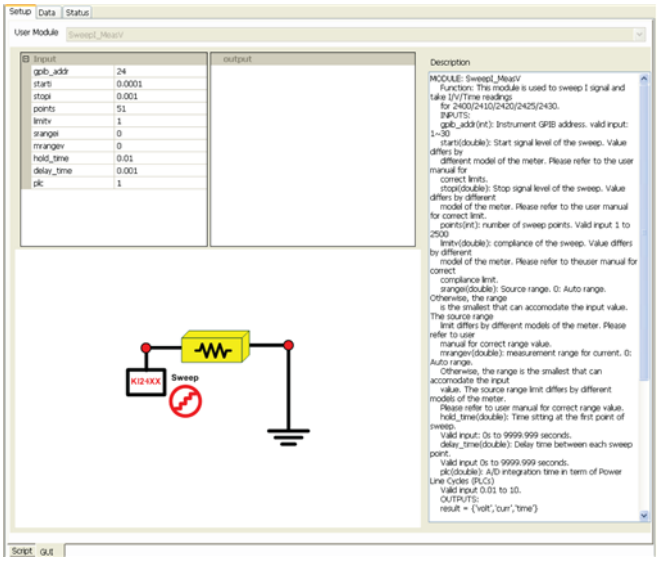
Inputs

`gpib_addr` (int):	Instrument GPIB address. Valid input: 1 through 30
`starti` (double):	Start the current sweep signal
`stopi` (double):	Stop the current sweep signal
`points` (int):	Number of sweep points (valid input 1 to 2500)
`sweepMode` (int):	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep.
`limitv` (double):	Voltage-sweep compliance value.
`srangei` (double):	Measurement range for voltage. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`mrangev` (double):	Measurement range for current. If zero is selected, the instrument will autorange. Otherwise, the range is the smallest that can accommodate the input value
`hold_time` (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 s
`delay_time` (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 s
`plc` (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.01 to 10)

Outputs

I		
V		
time		
Error:	Error value	
	0	OK
	-200	Instrument initialize error
	-300	Configuration error occurred
	-400	Reading error occurred
	-10000	(INVAL_INST_ID) The specified instrument ID does not exist
	-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred
	-10091	(GPIB_TIMEOUT) A timeout occurred during communications

Figure 91: Series 2400 instruments SweepI standard GUI



Configure a Model 622x and Model 2182A library

The following module will help you to configure your library of Keithley Instruments using the Model 662x and Model 2182A to perform delta tests to measure very low resistance.

Model 622x and Model 2182A delta test

To add a Model 622x and Model 2182A delta test module:

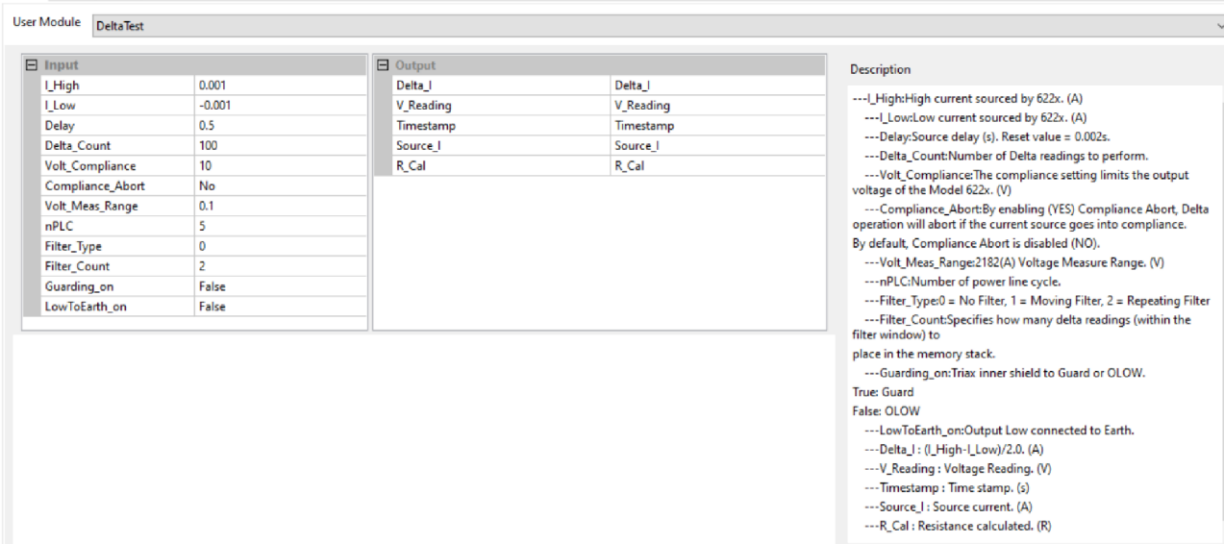
1. Add a PTM to the configuration navigator.
2. Import the `KI662x_2182_Lib.py` module from the PTMLib library. Choose the `DeltaTest` module from test module drop-down list.

This module is used to measure very low resistances with Model 622x and Model 2182A. A good method of doing this is called the delta method, with a Model 2182A sourcing the current and a Model 622x measuring the resistance.

Inputs	Description
`I_High`	High current sourced by 622x (A)
`I_Low`	Low current sourced by 622x (A)
`Delay`	Source delay (s) Reset value = 0.002s
`Delta_Count`	Number of Delta readings to perform
`Volt_Compliance`	The compliance setting limits the output voltage of the Model 622x (V)
`Compliance_Abort`	By enabling (YES) Compliance Abort, Delta operation will abort if the current source goes into compliance; By default, Compliance Abort is disabled (NO)
`Volt_Meas_Range`	2182(A) Voltage Measure Range (V)
`nPLC`	Number of power line cycles
`Filter_Type`	0 = No Filter, 1 = Moving Filter, 2 = Repeating Filter
`Filter_Count`	Specifies how many delta readings (within the filter window) to place in the memory stack
`Guarding_on`	Triaxial inner shield to Guard or OLOW
`LowToEarth_on`	Output Low connection
`Delta_I`	(I_High-I_Low)/2.0 (A)
`V_Reading`	Voltage Reading. (V)
`Timestamp`	Time stamp (s)
`Source_I`	Source current. (A)
`R_Cal`	Resistance calculated (R)

Outputs	Description
`Delta_I`	(I_High-I_Low)/2.0 (A)
`V_Reading`	Voltage Reading (V)
`Timestamp`	Time stamp (s)
`Source_I`	Source current (A)
`R_Cal`	Resistance calculated (R)

Figure 92: Delta test standard GUI



Model 622x and Model 2182A differential test

To add a Model 622x and Model 2182A differential test

1. Add a PTM to the configuration navigator
2. Import the `KI622x_2182_Lib.py` module from the PTMLib library. Choose the Differential Test module from test module drop-down list.

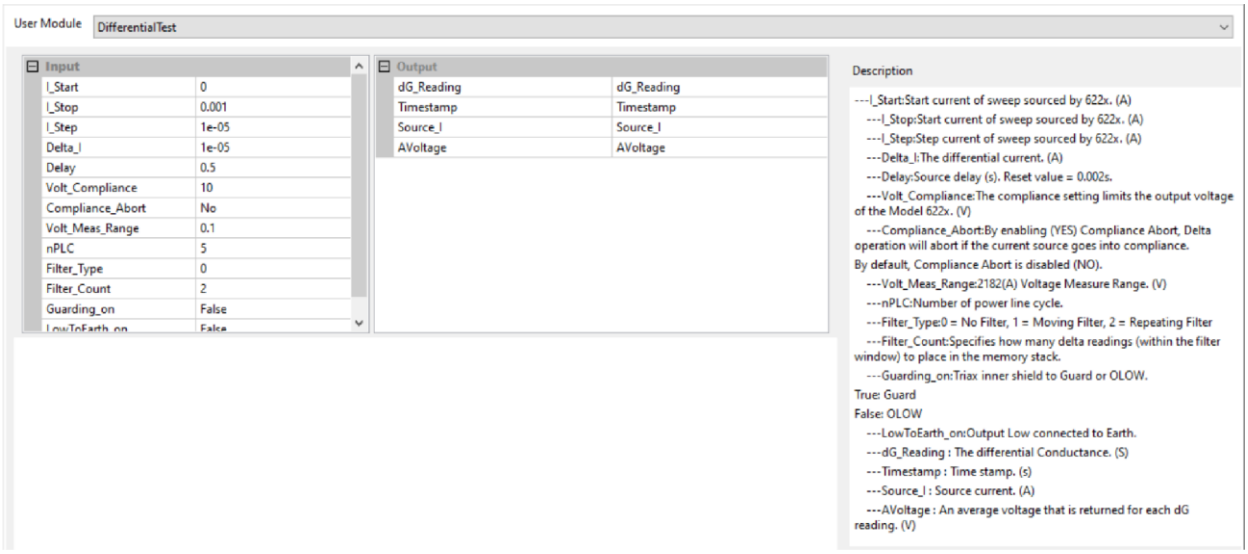
This module is used to measure differential conductance, with Model 2182A sourcing sweep current and Model 622x measure the differential conductance.

Inputs	Description
`I_Start:`	Start current of sweep sourced by 622x (A)
`I_Stop:`	Start current of sweep sourced by 622x (A)
`I_Step:`	Step current of sweep sourced by 622x (A)
`Delta_I:`	The differential current. (A)
`Delay:`	Source delay (s). Reset value = 0.002s
`Volt_Compliance:`	The compliance setting limits the output voltage of the Model 622x (V)
`Compliance_Abort:`	By enabling (YES) Compliance Abort, Delta operation will abort if the current source goes into compliance. By default, Compliance Abort is disabled (NO)
`Volt_Meas_Range`	2182(A) Voltage Measure Range (V)
`nPLC`	Number of power line cycles
`Filter_Type`	0 = No Filter, 1 = Moving Filter, 2 = Repeating Filter
`Filter_Count`	Specifies how many delta readings (within the filter window) to place in the memory stack
`Guarding_on`	Triaxial inner shield to Guard or OLOW

LowToEarth_on	Output Low connection

Outputs	Description
dG_Reading	Differential conductance (S)
Timestamp	Time stamp (s)
Source_I	Source current (A)
AVoltage	Average voltage that is returned for each dG reading (V)

Figure 93: Model 622x/2182A differential test



Configure a gate charge library

The following module helps you configure a gate charge library with a 4200A SMU to perform a gate charge test for power MOSFETs, based on the method in JEDEC standard JESD24-2, *Gate Charge Test Method*, available at jedec.org.

gatecharge test

To add a gatecharge module:

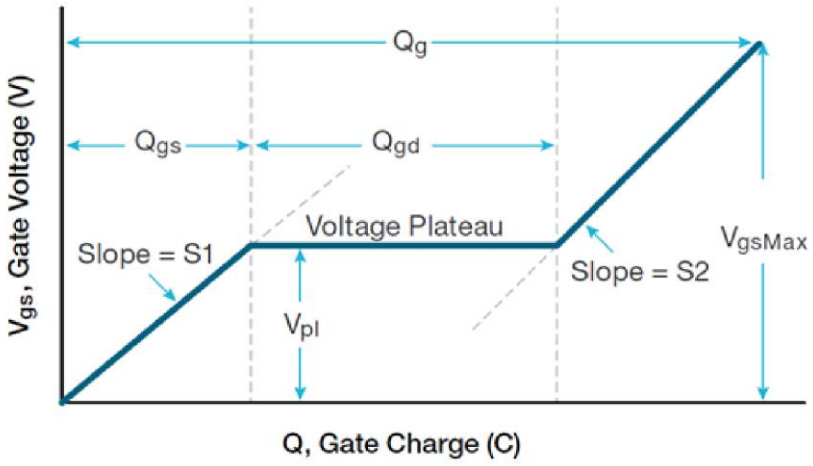
1. Add a PTM to the configuration navigator.
2. Import the `gateCharge.py` module from the PTMLib library.
3. Select the **gatecharge** module from test module drop-down list.

Required equipment: Keithley Instrument 4200A-SCS SMU.

This module performs a gate-charge test with the 4200A SMU. In this module, a fixed test current (Ig) is forced into the gate of a MOS transistor and the measured gate-source voltage (Vgs) is plotted against the charge flowing into the gate. A fixed voltage bias is applied to the drain terminal. From the resulting gate-voltage waveform, the gate-source charge (Qgs), gate-drain charge (Qgd), and gate charge (Qg) are derived.

The following figure shows a typical gate-voltage versus a gate-charge curve of a power MOSFET.

Figure 94: Typical gate voltage vs. gate charge curve of a power MOSFET

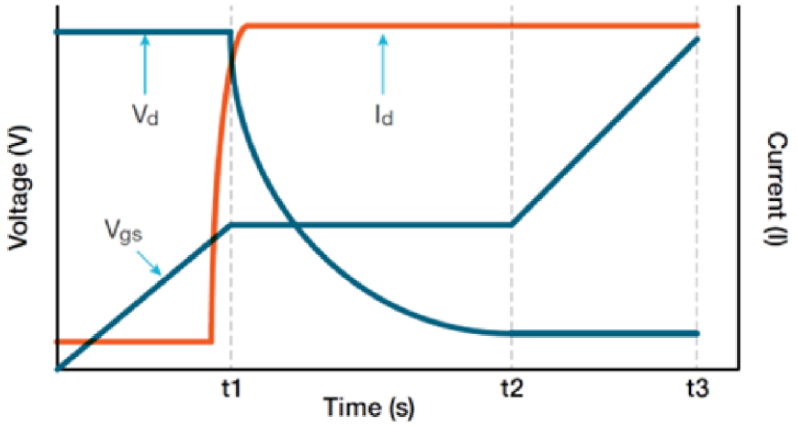


The gate-charge (Qg) is derived from the forced gate-current and time (Igdt). The gate-source charge (Qgs) is the charge required to reach the beginning of the plateau region where the gate-source voltage (Vgs) is almost constant (as shown in the following figure). The plateau voltage (Vpl) is defined, according to the JEDEC standards (jedec.org), as the gate-source voltage when `dVgs/dt` is at a minimum.

The voltage plateau is the region when the transistor is switching from the OFF state to the ON state. The gate-charge required to complete this switching is the same charge needed to switch the device from the start of the plateau region to the end. It is defined as gate-drain charge (Qgd) and is known as the *Miller* charge. The Qg is the charge from the origin to the point where the gate-source voltage (Vgs) is equal to a specified maximum (VgsMax).

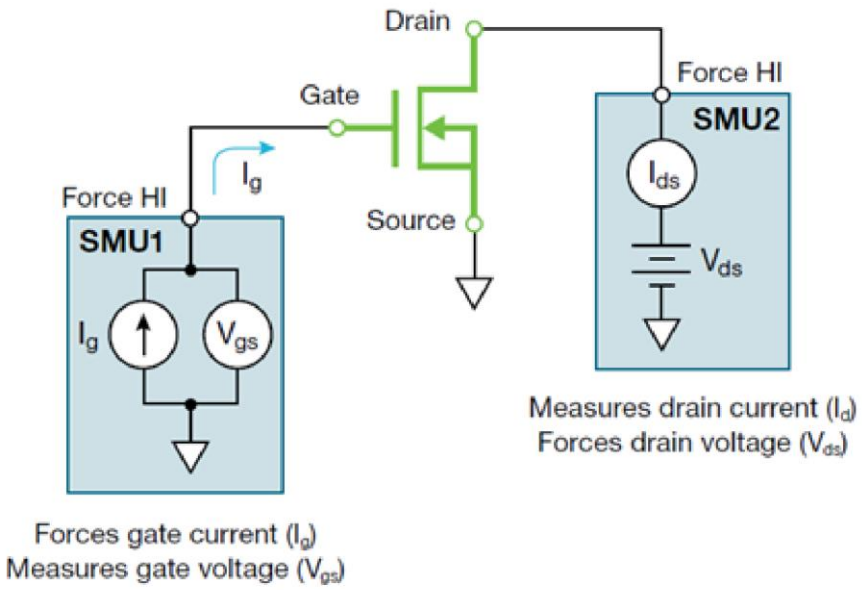
The following figure shows a typical gate and drain waveform as a function of time. As current is forced to the gate, the Vgs increases until it reaches the threshold voltage. At this point, the drain current (Id) begins to flow. When Cgs is charged up at time t1, the current drain (Id) stays constant and the drain voltage (Vd) decreases. The Vgs remains constant until it reaches the end of the plateau. Once the Cgd is charged at time t2, the gate-source voltage (Vgs) starts to increase again until it reaches the specified maximum gate-voltage (VgsMax).

Figure 95: Vgs, Vd, and Id vs. time of MOSFET



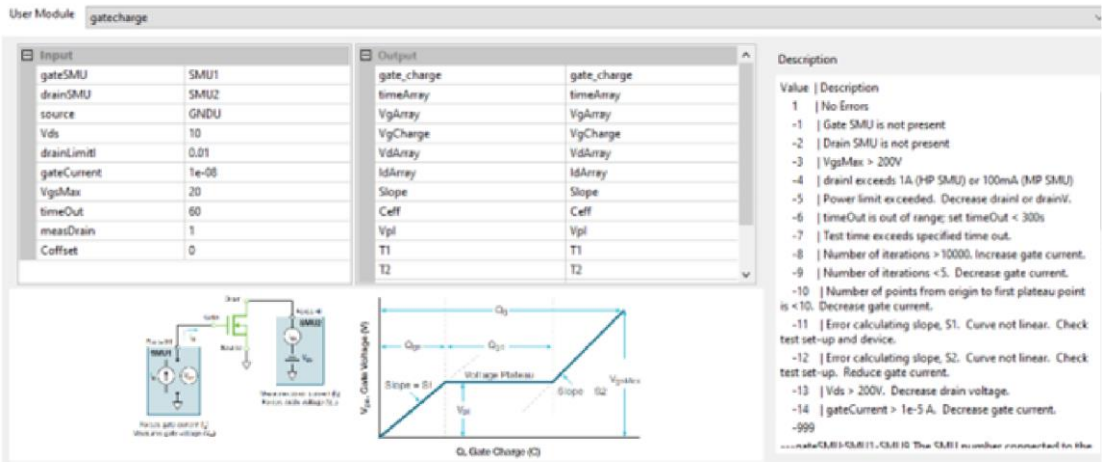
The 4200A-SCS measures the gate-charge of a power MOSFET using two SMU instruments. The following figure illustrates the basic circuit diagram of the gate-charge test. The Force HI terminal of one SMU (SMU1) is connected to the gate terminal of the MOSFET and forces the gate-current (Ig) and measures the gate-source voltage (Vgs) as a function of time. A second SMU (SMU2) applies a fixed voltage (Vds) to the drain at a specified current compliance (Ids). The MOSFET's source terminal is connected to the Force LO terminal or GNDU of the 4200A-SCS chassis.

Figure 96: Gate charge test using two SMU instruments



The following figure, and the input and output tables, indicate the input and output parameters for the gatecharge module and the standard GUI.

Figure 97: Gate charge standard GUI



Input	Description
gateSMU	SMU on gate (SMU1~SMU9)
drainSMU	SMU on drain (SMU1~SMU9)
source	The source terminal; GNDU as default
Vds	The drain bias voltage (V)
drainLimitI	Current compliance of the drain SMU
gateCurrent	The gate current (A)
VgsMax	The maximum voltage level of the gate SMU (V)
timeOut	The number of seconds prior to a time out (s)
measDrain	1: Return measured drain current 0: Not return measured drain current
Coffset	Run test with open circuit and enter the Ceff value returned in the Output data

Output	Description
gate_charge	Test status values
	Value | Description
	1 | No Errors
	-1 | Gate SMU is not present
	-2 | Drain SMU is not present
	-3 | VgsMax > 200V
	-4 | drainI exceeds 1A (HP SMU) or 100mA (MP SMU)
	-5 | Power limit exceeded; decrease drainI or drainV
	-6 | timeOut is out of range; set timeOut < 300s
	-7 | Test time exceeds specified time out
	-8 | Number of iterations >10000 Increase gate current
	-9 | Number of iterations <5; decrease gate current
	-10 | Number of points from origin to first plateau point is <10; decrease gate current
	-11 | Error calculating slope, S1; curve not linear; check test set-up and device
	-12 | Error calculating slope, S2; curve not linear; check test set-up; reduce gate current
	-13 | Vds > 200V; decrease drain voltage
	-14 | gateCurrent > 1e-5 A; decrease gate current
timeArray	Measured time (s)
VgArray	Measured gate-source voltage (V)
VgCharge	Measured gate charge (C)
VdArray	Measured drain voltage (V)
IdArray	Measured drain current (A)
Slope	Dynamic slope (dVg/dt) of gate voltage
Ceff	Ratio of gate charge to maximum gate voltage
Vpl	Plateau or Miller voltage (V)
T1	Timestamp where the plateau area begins (s)
T2	Timestamp where the plateau area ends (s)
Qgs	Gate charge from the origin to the first inflection point, or the voltage plateau (C)
Qgd	Gate charge between the two inflection points in the gate charge curve (C)
Qg	Gate charge from the origin to VgsMax (C)

Correcting for Offset Capacitances

Depending on the cabling and connections of the measurement system, the offset capacitance can be in the single picofarads to hundreds of picofarads ranges. These capacitances can be corrected by executing the gate-charge module with an open circuit, obtaining the offset capacitance, then entering the offset capacitance value back in the gate-charge module as the input parameter `Coffset` for the compensation of the subsequent tests.

1. To measure the offset capacitance:

To measure the offset capacitance, set up the test parameters, including the input gate current as though the device were connected to the SMUs. However, increase the VgsMax just for the Ceff measurement. Prior to executing the test, lift the probes, or remove the device from the test fixture. Execute the gate-charge test with an open circuit.

2. To obtain the offset capacitance:

To obtain the offset capacitance, after the test is executed, the measured offset capacitance of the system is calculated and appears in the Ceff column in the data panel. Ceff is derived from the maximum gate voltage, gate current, and time. Because an open circuit is measured during this step, a test status value of –9 or –12 may appear in the data panel after the test is executed. This is because no device is measured, so there is no plateau region. However, the Ceff value is correct and can be entered as the input parameter Coffset in the GUI the of gate-charge module.

3. To enter the measured offset capacitance and execute:

Enter the value of the measured offset capacitance Ceff as the input parameter Coffset on the GUI of gate-charge module. By default, Coffset is 0 F. Compensation will be made based on the offset capacitance Coffset in the subsequent runs of the gate-charge module.

Section 6

UAP and global variable definitions

In this section:

Overview

User access points (UAPs) allow you to extend the features and functions of the Keithley-provided test execution engine at the test, device, subsite, site, wafer, and cassette (lot) levels. This is accomplished by executing the Python test module (PTM), the C-language test module (CTM), or the script test module (STM) at the entry or exit of each level of automation within ACS.

ACS global variables and functions are used in a UAP routine to get test information, to control the testing process, or to write custom data file results.

This document describes the global data and global functions that are available in ACS and provides some application examples. All these descriptions and examples are based on PTMs.

Global variables

Global data variables can be accessed from a user access point (UAP) Python test module (PTM). All global variables start with the characters `ACS_xxxx`. The following table lists the ACS global variables.

ACS global variables

Variable name	Type	Comment	UAP level	Example
`ACS_frame`	string	The frame of ACS	Any UAP	Object
`ACS_uap_level`	string	Present UAP level	Any UAP	`wafer_begin`, `site_end`
`ACS_proj_path`	string	Present path	Any UAP	`C:\ACS\ Projects\default`
`ACS_proj_name`	string	Present project name	Any UAP	Default
`ACS_lot_id`	string	Present lot ID	Any UAP except lot begin	`lotid001` (if single wafer)
`ACS_slot_no`	int	Present slot number	Wafer level and below	`1`, `2`, ...`25`
`ACS_wafer_id`	string	Present wafer ID	Wafer level and below	`waferid001`
`ACS_pattern_id`	string	Present pattern ID	Pattern level and below	`Pattern1`
`ACS_site_id`	string	Present die ID	Site level and below	`Site_n1p1`
`ACS_site_coord`	tuple	Present die coordinates	Subsite level and below	`(-1, 1)`
`ACS_ssite_id`	string	Present subsite ID	Subsite level and below	`subsite1`
`ACS_ssite_loop`	int	Present subsite index of subsite loop	Subsite level and below	`1`
`ACS_ssite_loopNum`	int	Total loop number of the present subsite	Subsite level and below	`1`
`ACS_ssite_coord`	tuple	Present subsite coordinates	Subsite level and below	`(0.0,0.0)`
`ACS_dut_id`	string	Present DUT ID	DUT level and below	Resistor
`ACS_test_id`	string	Present test ID	Test level	`sweepv`
`ACS_test_data`	dictionary	Present test data	Test end level	See Variable data type examples (on page 6-5)
`ACS_wdf_head`	dictionary	WDF header information	Wafer level and below	See Variable data type examples (on page 6-5)
`ACS_wdf_nonexist`	dictionary	WDF erased sites dictionary	Any UAP	See Variable data type examples (on page 6-5)
`ACS_t_start`	string	Test start time	Any UAP	`08-Aug-2008 20:08`
`ACS_t_end`	string	Test end time	Any UAP	`08-Aug-2008 20:10`

ACS global variables

Variable name	Type	Comment	UAP level	Example
ACS_first_wafer	bool	Is this the first wafer in the cassette?	Wafer level and below	True or False
ACS_last_wafer	bool	Is this the last wafer in the cassette?	Wafer level and below	True or False
ACS_first_site	bool	Is this the first site of the wafer?	Site level and below	True or False
ACS_last_site	bool	Is this the last site of the wafer?	Site level and below	True or False
ACS_output_list	list	All tests output list	Any UAP	See Variable data type examples (on page 6-5)
ACS_temp_klf	string	Temporary klf file location	Any UAP	C:\DOCUME~1\kiadmin\LOCALS~1\Temp
ACS_prb_errcode	int	Prober error code	Any UAP	0, -1013
ACS_operator	string	Operator field from automation panel	Any UAP, from Automation	Same as entry
ACS_die_type	string	Die type from automation panel	Any UAP, from Automation	Same as entry
ACS_remark	string	Remark from automation panel	Any UAP, from Automation	Same as entry
ACS_session_name	string	Session name from automation panel	Any UAP, from Automation	Same as entry
ACS_equipment_id	string	Equipment from automation panel	Any UAP, from Automation	Same as entry
ACS_fixture_id	string	Fixture ID from automation panel	Any UAP, from Automation	Same as entry
ACS_testplan_ver	string	Test Plan Version from automation panel	Any UAP, from Automation	Same as entry
ACS_process_level	string	Test Process Level from automation panel	Any UAP, from Automation	Same as entry
ACS_userdata_path	string	Full path of user data	Any UAP	C:\ACS\user_data
ACS_ptm_path	string	PTM directory setting in preference	Any UAP	C:\ACS\library\pyLibrary\PTMLib
ACS_site_passfail_info	string	Did the present site pass or fail?	Any UAP	pass or fail
ACS_test_wafer_ids	list	Save the IDs of the tested wafers	Any UAP, from Automation	[01, 02, 03, 04]

ACS global variables

Variable name	Type	Comment	UAP level	Example
`ACS_rpt_each_wafer`	int	Create a kdf file for each wafer? (1= yes, 0 = no)	Any UAP	`0`
`ACS_ktxe_exit`	int	Stop the execution of ktxe	Any UAP	`1`
`ACS_ktxe_loop_wafer`	int	Present wafer loop flag	Any UAP except for lot end	`1`
`ACS_run_mode`	string	Indicate automation or manual run	Any UAP	`automation` or `manual`
`ACS_Disable_Subsite_List`	list	If the subsite name is in this list, the subsite is skipped	Test end level	`[pre_test]`
`ACS_Disable_DUT_List`	list	If the subsite name is in this list, the subsite is skipped	Test end level	`['DEV']`
`ACS_Disable_Test_List`	list	If the subsite name is in this list, the subsite is skipped	Test end level	`[MOS]`
`ACS_Stress_Time`	float	The accumulated time for the measure stress loop; changed to different value according to subsite loop number	Any UAP	`10`
`ACS_Stress_Duration`	float	Stress duration time for each subsite loop	Any UAP	`5`

NOTE

Most global variables are available at any UAP level; however, the value is accurate only when it is accessed at the level documented in the UAP Level column in the previous table.

You can access each of these variables directly by name. You can view the variable values using the `logMessage` function, which is defined in `LogManager.py`. You can view the messages in the log window at the bottom of the ACS user interface.

If the variable type is string, you can view its value using the following code in a UAP routine:

```
import LogManager as log
log.logMessage(ACS_lot_id)
```

If the variable type is not string, you need to add the `str()` function to change the variable type to string.

```
import LogManager as log
log.logMessage(str(ACS_test_data))
```

See Variable data type examples (on page 6-5) for examples of some of the more complicated variable data types.

Variable data type examples

The following topics describe some of the more complicated variable data types.

ACS_output_list

Type: List

Structure:

```
[outputName@testID, …]
```

Example:

```
['V_Pos@itm', 'I_Pos@itm']
```

ACS_test_data

Type: Dictionary

Structure:

```
{
'GROUP1' : {var1: […], var2: […],…},
'GROUP2' : { var1: […], var2: […],…},
...
}
```

Example:

```
{
'GROUP1': {'V_Pos': [0.39491547641171859],
'I_Pos': [0.16948805017700203]
}
}
```

NOTE

`ACS_test_data` is assigned when the test ends, and the test data resets to `{}` when the test begins.

ACS_wdf_nonexist

Type: Dictionary

Structure:

```
{(x, y):Site_pxny, …}
```

Example:

```
{(1, 3): 'Site_p1p3', (3, 3): 'Site_p3p3'}
```

ACS_wdf_head

Type: Dictionary

Structure:

```
{
            'version' : string version,
               'file' : string filename,
               'date' : string date,
               'comment' : string comment,
               'project' : string Single,
               'diameterunits' : string Metric|English,
               'diameter' : float diameter,
               'units' : string Metric|English,
               'squarewaferx' : float width, wafer width in square wafer,
               'squarewafery' : float height, wafer height in square wafer,
               'diesizex' : float diesizex, or Glass width in LCD,
               'diesizey' : float diesizey, or Glass length in LCD,
               'orientation' : (string Flat|Notch,
                           string Top|Bottom|Left|Right,
                           string Top|Bottom|Left|Right),
            'waferoffset' : (float xoffset, float yoffset),
            'axis' : int 1|2|3|4,
       'origin' : (int xdistance_to_target,int ydistance_to_target),
            'target' : (int targetx, int targety),
            'autoalignlocation' : (float alignx, float aligny),
            'optimize' : int 0-9,
            'siteusage' : int usage,
            'chipsperreticle' : (float, float),
            'reticletarget' : (float, float), or Mark1 position in LCD
            'reticleoffset' : (float, float), or Mark2 position in LCD
       'partial': int 0-1, or whether to show coordinate on glass map in LCD
            'margin': float,
        'color': {'default': string, 'margin': string, 'target': string, 'target
   selected':   string,
          'current': string, 'touched': string, 'pass': string, 'fail':
                string},
            'numx': int Site num in x direction, added for LCD
            'numy': int Site num in y direction, added for LCD
            'squareflag': int 0-1, default 0
            'userdefvalue1' : string,
            'userdefvalue2' : string,
                :
                :
            'userdefvalue10' : string
        }
```

Example:

```
          {'comment': '',
'diameter': 8.0,
'partial': 0,
'orientation':('Flat','Left','Bottom'),
'squareflag': 0,
'color': {'targetsel': (255, 255, 0),
 'target': (255, 0, 0),
```

```
'default': (129, 243, 235),
'invalid': (193, 193, 193),
'current': (0, 0, 255),
'grid': (157, 149, 130),
'touched': (0, 255, 0),
'pass': (0, 255, 0),
'fail': (255, 0, 0),
'margin': (171, 168, 217)},
'optimizepriority': 'die',
'userdefvalue10': '',
'file': '',
'axis': 2,
'diameterunits':'English',
'scale': 1,
'autoalignlocation': (0.0, 0.0),
'siteusage': 0,
'userdefvalue6': '',
'userdefvalue7': '',
'userdefvalue4': '',
'userdefvalue5': '',
'userdefvalue2': '',
'userdefvalue3': '',
'version': 1.1000000000000001,
'userdefvalue1': '',
'units': 'Metric',
'reticleoffset': (0.0, 0.0),
'userdefvalue8': '',
'userdefvalue9': '',
'logicrow': 1,
'numx': 4,
'numy': 4,
'diesizey': 20.0,
'logiclist': [(0, 0)],
'reticletarget': (0.0, 0.0),
'date': '',
'diesizex': 20.0,
'optimize': 0,
'origin': (5, 5),
'movestepdiv': 10,
'target': (0, 0),
'solid': 1,
'waferoffset': (0.0, 0.0),
'project': 'Single',
'logiccol': 1,
'chipsperreticle': (0.0, 0.0),
'squarewaferx': 100.0,
'squarewafery': 100.0,
'margin': 2}
```

ACS_temp_klf

Type: String

Usage: Defines the location of a temporary limits file. You can construct a Keithley limits file (`.klf`) object and get limits information. The `.klf` object has two common properties: head and limits. Head is dictionary type and limit is a list type.

Head structure of .klf:

```
    {
        'version' : string,     # Version of the limits file
        'file'    : string,     # Name of the limits file
        'date'    : string,     # Update date of the limits file
        'comment' : string      # Comments
    }
```

Limits structure of .klf:

```
   [{
        'id'  : string,         # ID of result
        'nam' : string,         # Friendly name of result
        'unt' : string,         # Unit of result
        'rpt' : int,            # Report flag, 0/1
        'crt' : int,            # Critical level, 0~9
        'tar' : float,          # Target value
        'val' : (float, float),    # Valid limits
        'spc' : (float, float),    # Spec limits
        'cnt' : (float, float),    # Control limits
        'eng' : (float, float),    # Engineer limits
        'af'  : int,            # Abort limit
        'al'  : int,            # Abort level
        'ena' : int             # Enable/Disable (requires adaptive test)
},
…
]
```

Example:

```
import LogManager as log
import KLF

klf = KLF.KLF(ACS_temp_klf)
log.logMessage(str(klf.head))
log.logMessage(str(klf.limits))
```

Other usage examples

There are usage examples for the `ACS_Stress_Time`, `ACS_Stress_Duration`, `ACS_Disable_Subsite_List`, `ACS_ Disable_DUT_List`, and `ACS_Disable_Test_List` variables in the following projects:

- DC stress-measure loop project (in `C:\ACS\Projects\DC_Stress_Measure_loop`).
- Pulse stress-measure loop project (in `C:\ACS\Projects\Pulse_Stress_Measure_Loop`).

NOTE

The pulse stress-measure loop project is only available when ACS is installed on the Model 4200-SCS.

ACS global functions

Functions that can be accessed in the user access point (UAP) Python test modules (PTMs) are listed in the following table.

Function type	Function name	Comment
User global data functions	FUNC_SET_GLOBAL_VALUE(name, value) (on page 6-12)	Update global variable with a value; only this function can change a global variable value
	FUNC_GET_GLOBAL_VALUE(name) (on page 6-12)	Get the value of a global variable
	FUNC_SET_USER_GLOBAL(name) (on page 6-12)	Define a new variable in user global data pool; only used between UAPs
	FUNC_SYS_GLOBAL(name, value) (on page 6-13)	Set a system global variable value
Execution engine related functions	FUNC_EXIT_KTXE() (on page 6-14)	Exit Keithley Test Execution Engine (KTXE) execution
	FUNC_SET_KTXE_SUBSITES(subsitelist) (on page 6-14)	Set subsites list used by the execution engine
	FUNC_SET_KTXE_LOOP_WAFER (on page 6-13)	Set wafer loop flag
	FUNC_EXEC_TEST() (on page 6-22)	Execute the specified test module
KDF manipulation related functions	FUNC_GET_KDF_HEADER() (on page 6-14)	Get the Keithley data file (`.kdf`) header; returns data as a dictionary
	FUNC_SAVE_KDF_HEADER() (on page 6-15)	Save the Keithley data file header dictionary into a `.kdf` file
	FUNC_SAVE_KDF_WAFERID(kdf_file, wafer_id) (on page 6-16)	Save Keithley data file wafer ID into a `.kdf` file
	FUNC_SAVE_KDF_EOW(kdf_file) (on page 6-16)	Save Keithley data file wafer delimiter (<EOW>) into a `.kdf` file
	FUNC_SAVE_KDF_SITEID(kdf_file, site_id, site_x, site_y) (on page 6-17)	Save the Keithley data file site ID into a `.kdf` file
	FUNC_SAVE_KDF_EOS() (on page 6-17)	Save the Keithley data file site delimiter (<EOS>) into a `.kdf` file
	FUNC_GET_KDF_DATA(wafer_id, site_id) (on page 6-18)	Get the `.kdf` data for the specified wafer and site ID
	FUNC_GET_NEW_KDF_DATA(wafer_id, site_id) (on page 6-19)	Get the latest `.kdf` data for the specified wafer and site id
	FUNC_SAVE_KDF_DATA() (on page 6-18)	Save Keithley data file data into a `.kdf` file
	FUNC_SITE_COORD_2_ID(coord) (on page 6-20)	Translate a site coordinate to site ID
	FUNC_SITE_ID_2_COORD(coord)	Translate a site ID to site coordinate
Get object functions	FUNC_GET_WDF_OBJ() (on page 6-21)	Get the wafer definition file (WDF) object used by the execution engine
	FUNC_GET_KDF_OBJ(dut_name) (on page 6-28)	Get the Keithley data file (KDF) object from the test tree used by the execution engine according to the name of the DUT according to the name of the DUT
	FUNC_GET_SUBSITE_OBJ(subsite_name) (on page 6-25)	Get the specified subsite object from the test tree used by the execution engine according to the name of the DUT
	FUNC_GET_DUT_OBJ(dut_name) (on page 6-26)	Get the specified device under test (DUT) object from the test tree used by the execution engine according to the name of the DUT
	FUNC_GET_TEST_OBJ(test_name) (on page 6-24)	Get the specified test object from the test tree

Function type	Function name	Comment
Get test info functions	GET_EXEC_TEST_DIC() (on page 6-22)	Get the content of the test dictionary
	GET_EXEC_TEST_LIST() (on page 6-23)	Get a test list; it is the key in the test dictionary
	FUNC_GET_DUT_NAME(test_name) (on page 6-27)	Get the name of the DUT

FUNC_SET_GLOBAL_VALUE(name, value)

This function updates a specified global variable with a specified value. This function is the only function that can change the value of a global variable.

Usage

```
FUNC_SET_GLOBAL_VALUE(name, value)
```

name	String; the name of the global variable to change; see Global variables (on page 6-2) for a list of values
value	String; the value of the global variable

Example

```
FUNC_SET_GLOBAL_VALUE('ACS_wafer_id', 'wafer_1')
```

This example updates the `ACS_wafer_id` global variable to `wafer_1`.

NOTE

A variable must have quotation marks, for example: `'ACS_wafer_id'` or `'wafer_1'`. This function can update the value of the corresponding global variable, but it cannot update the value saved in the `.kdf` file. If you want the updated value stored in the `.kdf` file, use the KDF Manipulation Related Functions listed in ACS Global functions (on page 6-11).

FUNC_GET_GLOBAL_VALUE(name)

This function gets the value of the global variable; you can use the name of the variable directly.

Usage

name	String; the name of the global variable to change; see Global variables (on page 6-2) for a list of values

Returns

A string

Example

```
FUNC_GET_GLOBAL_VALUE('ACS_wafer_id')
Or ACS_wafer_id
```

FUNC_SET_USER_GLOBAL(name)

This function adds a global user variable to the data pool. After the variable is set, this global user variable can be used by name directly at the present UAP and below.

Usage

```
FUNC_SET_USER_GLOBAL(name)
```

name	String; the name of the global variable to change; see Global variables (on page 6-2) for a list of values

Example

```
g_site_id_list = []
# g_site_id_list must be declared in advance
FUNC_SET_USER_GLOBAL('g_site_id_list ')
```

FUNC_SET_KTXE_LOOP_WAFER()

This function sets the `ACS_ktxe_loop_wafer` global variable, which is the loop flag for the present wafer.

Usage

```
FUNC_SET_KTXE_LOOP_WAFER (value)
```

value	Integer: `1` or `0`

Details

If the value is `1`, the present wafer is tested repeatedly until the flag is set to `0`. This function is only effective in single-wafer mode.

Example

```
FUNC_SET_KTXE_LOOP_WAFER(1)
```

This example sets the present wafer to be tested repeatedly.

FUNC_SYS_GLOBAL(name, value)

This function sets the value of the `a` system global variable.

Usage

```
FUNC_SYS_GLOBAL(name, value)
```

name	String; the name of the system global variable to change; see Global variables (on page 6-2) for a list of values for system global variables
value	String; the value of the system global variable you are changing

Example

```
FUNC_SYS_GLOBAL('ACS_wafer_id', 'wafer_1')
```

This example sets the value of the `ACS_wafer_id` system global variable to Wafer 1.

FUNC_EXIT_KTXE()

This function exits the execution engine.

Usage

```
FUNC_EXIT_KTXE()
```

Example

```
FUNC_EXIT_KTXE()
```

FUNC_SET_KTXE_SUBSITES(subsitelist)

This function changes the subsite list in the execution engine. Use on the site-level user access point (UAP) and below.

Usage

```
FUNC_SET_KTXE_SUBSITES(subsitelist)
```

subsitelist	List; format: `[(x1,y1), (x2, y2), …]`

Example

```
FUNC_SET_KTXE_SUBSITES([(0.0,0.1),(0.2,0.3)])
```

FUNC_GET_KDF_HEADER()

This function gets the Keithley data file (`.kdf`) header. Use this function at the beginning of the UAP lot.

Usage

```
FUNC_GET_KDF_HEADER()
```

Returns

- Dictionary, if a `.kdf` exists
- `False`, if a `.kdf` does not exist

Return structure:

```
{
'typ'   : string type,
'lot'   : string lot ID,
'prc'   : string process ID,
'dev'   : string device ID,
'tst'   : string test program name,
'sys'   : string tester name,
'tsn'   : int tester number,
'opr'   : string operator name,
'stt'   : string start time,
'sk1'   : string search key 1,
```

```
'sk2'   : string search key 2,
'sk3'   : string search key 3,
'lmt'   : string limit file name,
'wdf'   : string wafer description file name,
'com'   : comment
}
```

Example

```
FUNC_GET_KDF_HEADER ()
```

Example return data:

```
{
'tsn': 1,
'stt': '26-Feb-2019 10:55',
'opr': 'wendy',
'prc': '1',
'lmt': '',
'dev': 'exampledietype',
'sys': 'WENDY',
'lot': 'lotid',
'tst': 'Keithley S500',
'com': 'exampleremark',
'wdf': ''
}
```

FUNC_SAVE_KDF_HEADER()

This function saves the Keithley data file (kdf) header dictionary into a corresponding `.kdf` file.

Usage

`FUNC_SAVE_KDF_HEADER(`*`kdf_file`*`, `*`header`*`)`

kdf_file	String; specifies the full name of the `.kdf` file
header	Dictionary; kdf header information

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_HEADER('C://a.kdf', {'tsn': 1, 'stt': '26-Feb-2009 10:55', 'opr':
   'wendy', 'prc': '1', 'lmt': '', 'dev': 'exampledietype', 'sys': 'WENDY',
   'lot': 'lotid', 'tst': 'Keithley S500', 'com': 'exampleremark', 'wdf': ''})
```

FUNC_SAVE_KDF_WAFERID(kdf_file, wafer_id)

This function saves the kdf wafer ID into a corresponding `.kdf` file.

Usage

```
FUNC_SAVE_KDF_WAFERID(kdf_file, wafer_id)
```

kdf_file	String; the full name of the `.kdf` file
wafer_id	String; the ID of the wafer

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_WAFERID('C://a.kdf', '01')
```

FUNC_SAVE_KDF_EOW(kdf_file)

This function saves the kdf wafer delimiter (<EOW>) into a `.kdf` file.

Usage

```
FUNC_SAVE_KDF_EOW(kdf_file)
```

kdf_file	String; specifies the full name of the `.kdf` file

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_EOW('C://a.kdf')
```

This example saves the kdf wafer delimiter to a kdf file named `a.kdf`.

FUNC_SAVE_KDF_SITEID(kdf_file, site_id, site_x, site_y)

This function saves the kdf site ID into a `.kdf` file.

Usage

`FUNC_SAVE_KDF_SITEID(`*`kdf_file`*`, `*`site_id`*`, `*`site_x`*`, `*`site_y`*`)`

kdf_file	String; specifies the full name of the `.kdf` file
site_id	String; ID of the site to save
site_x	String; x-coordinate of the site
site_y	String; y-coordinate of the site

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_SITEID('c:\\a.kdf', 'Site_p4p5', '4','5')
```

This example saves site p4p5 coordinates to a kdf file named `a.kdf`.

FUNC_SAVE_KDF_EOS()

This function saves the kdf site delimiter (<EOS>) into a `.kdf` file.

Usage

`FUNC_SAVE_KDF_EOS(`*`kdf_file`*`)`

kdf_file	String; specifies the full name of the `.kdf` file

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_EOS('c:\\a.kdf')
```

This example saves kdf site delimiter into a file named `a.kdf`.

FUNC_SAVE_KDF_DATA()

This function saves the specified kdf data into a `.kdf` file.

Usage

```
FUNC_SAVE_KDF_DATA(kdf_file, data)
```

kdf_file	String; specifies the full name of the `.kdf` file
data	List; the data to be written into the `.kdf` file

Returns

- `True`: If write to file was successful
- `False`: If write to file was not successful

Example

```
FUNC_SAVE_KDF_DATA('c:\\a.kdf', [['I_Pos@itm@HOME#1@GROUP1',
   0.033643178212167946], ['I_Pos@itm@HOME#2@GROUP1', 0.44579186631844625]])
```

FUNC_GET_KDF_DATA(wafer_id, site_id)

This function gets the kdf data for the specified wafer and site id.

Usage

```
FUNC_GET_KDF_DATA(wafer_id, site_id)
```

wafer_id	String; ID of the wafer
site_id	String; ID of the site

Returns

- List of data if the kdf data file was read successfully.
- `False` if the kdf data file was not read successfuly.

Return structure:

```
[[padname@testname@subsitename#subsitlistid@groupname],…]
```

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_KDF_DATA(ACS_wafer_id, ACS_site_id)))
```

Example output:

```
[['I_Pos@itm@HOME#1@GROUP1', 0.033643178212167946], ['I_Pos@itm@HOME#2@GROUP1',
0.44579186631844625], ['I_Pos@itm_2@HOME_2#1@GROUP1', 0.36503035785604399],
['I_Pos@itm_2@HOME_2#2@GROUP1', 0.012356422404429968]]
[['V_Pos@sweepv@subsite', [0.002973165938638308, 0.0059463318772766159,
0.0089194978159149244, 0.011892663754553232, 0.014865829693191539]]]
```

FUNC_GET_NEW_KDF_DATA(wafer_id, site_id)

This function gets the latest Keithley data file (`.kdf`) data for the specified wafer and site id when the site priority is pattern first and the same site is selected in multiple patterns.

Usage

wafer_id: string, ID of wafer.

site_id: string, ID of site.

Details

This function differs from the `FUNC_GET_KDF_DATA` function only when the site sequence priority is pattern first and the same site is selected in multiple patterns.

Returns

- A list if the kdf data was read successfully
- `False` if the kdf file was not read successfully

Return structure:

```
[[padname@testname@subsitename#subsitlistid@groupname],…]
```

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_KDF_DATA(ACS_wafer_id, ACS_site_id)))
log.logMessage(str(FUNC_GET_NEW_KDF_DATA(ACS_wafer_id, ACS_site_id)))

#Line OUTPUT:
[['I_Pos@itm@HOME#1@GROUP1', 0.033643178212167946], ['I_Pos@itm@HOME#2@GROUP1',
   0.44579186631844625], ['I_Pos@itm_2@HOME_2#1@GROUP1', 0.36503035785604399],
   ['I_Pos@itm_2@HOME_2#2@GROUP1', 0.0123564224044299681]

#Line OUTPUT:
[['I_Pos@itm_2@HOME_2#1@GROUP1', 0.36503035785604399],
   ['I_Pos@itm_2@HOME_2#2@GROUP1', 0.0123564224044299681]
```

Build a project with two patterns and select the same site in both patterns. Work in single mode. Execute the code above at the site end UAP for an example of how to use this function.

FUNC_SITE_COORD_2_ID(coord)

This function translates the site coordinate to a site ID.

Usage

`FUNC_SITE_COORD_2_ID(`*`coord`*`)`

coord	Tuple; the site coordinates to use for the site ID

Returns

- A string if the translation was successful, in the form of `Site_p1n2`.
- `None` if the translation was not successful.

Example

```
import LogManager as log
log.logMessage(str(FUNC_SITE_COORD_2_ID((1,-2))))
```

This example translates the specified coordinates to a site ID name of `Site_p1n2`.

FUNC_SITE_ID_2_COORD(coord)

This function translates the site id into site coordinates.

Usage

`FUNC_SITE_ID_2_COORD(`*`siteid`*`)`

siteid	String; in the form of `Site_p1n2`

Returns

- Coord: Tuple; site coordinates in the form of `(1,-2)`
- `None` if the translation was not successful

Example

```
import LogManager as log
log.logMessage(str(FUNC_SITE_ID_2_COORD(Site_p1n2)))
```

This example translates the `Site_p1n2` ID to the coordinates (1, –2).

FUNC_GET_WDF_OBJ()

This function gets the wdf object used by the execution engine.

Usage

```
FUNC_GET_WDF_OBJ().wdf_obj
```

.wdf_obj	The wdf object property (see Details for a list of properties)

Details

Use the wdf object called by this function to get properties and to be able to call functions. All properties and functions can be called using the period (`.`) operator after getting the wdf object.

WDF object properties

- `self.head` (same as `ACS_wdf_head`)
- `self.patterns`
- `self.subsites`
- `self.touchedsites`
- `self.passsites`
- `self.failsites`
- `self.currentsite`
- `self.finishsite` (for LCD)
- `self.nonexist` (the nonexistent dice defined by user)
- `self.diemapping`

WDF object operations

- `load(wdf_file)`: Loads the `.wdf` file into a wdf object
- `save(wdf_file)`: Saves the wdf object to `.wdf` file
- `clear()`: Clears the wdf object
- `reset()`: Resets all wdf object properties to default values

Returns

- The `wdf_object` if the wdf object is in the execution engine
- `None` if there is no wdf object in the execution engine

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_WDF_OBJ().head))
FUNC_GET_WDF_OBJ().clear()
```

GET_EXEC_TEST_DIC()

This function gets the contents of the test dictionary.

Usage

```
GET_EXEC_TEST_DIC()
```

Returns

- `test_dict`: Dictionary; site coordinates in the form of `(1,-2)`
- `None` if there is a failure to extract contents of the test dictionary

Return structure

```
{(waferid, pattern_name, site_id, subsite_name, device_name, test_name):
   test_Object, …}
```

Example

```
import LogManager as log
log.logMessage(str(GET_EXEC_TEST_DIC()))
```

Example output:

```
{('01', 'Pattern_1', 'Site_p2p1', 'HOME', 'device_26', 'itm'): <testtree.TEST
instance at 0x063BBFD0>, ('01', 'Pattern_1', 'Site_p2p1', 'HOME', 'device_26',
'itm_1'): <testtree.TEST instance at 0x063D87B0>}
```

FUNC_EXEC_TEST()

This function executes a test.

Usage

```
FUNC_EXEC_TEST(location)
```

location	Tuple; format: ('*waferid*', '*pattern_name*', '*site_id*', '*subsite_name*', '*device_name*', '*test_name*')

Returns

- Test data, dictionary
- `None` if execution was unsuccessful

Example

```
import LogManager as log
log.logMessage(str(FUNC_EXEC_TEST(('03', 'Pattern_1', 'Site_n2p2', 'subsite',
    'Resistor_1k', 'sweepv'))))
```

Example output:

```
{ 'V_Pos':[0.12319012835262809,0.24638025670525618,
0.36957038505788425,0.49276051341051236,
0.61595064176314041],
'I_Pos': [0.153436046185253, 0.306872092370506, 0.46030813855575897,
0.613744184741012, 0.76718023092626497]}
```

GET_EXEC_TEST_LIST()

This function gets the execution test list.

Usage

```
GET_EXEC_TEST_LIST()
```

Returns

- `test_list`: List
- `None` if the operation fails.

Return structure

```
[(waferid, pattern_name, site_id, subsite_name, device_name, test_name),…]
```

Example

```
import LogManager as log
log.logMessage(str(GET_EXEC_TEST_LIST()))
```

Example output:

```
[('01', 'Pattern_1', 'Site_p2p1', 'HOME#1', 'device_26', 'itm'), ('01',
'Pattern_1', 'Site_p2p1', 'HOME#1', 'device_26', 'itm_1'), ('01', 'Pattern_1',
'Site_p2p1', 'HOME#2', 'device_26', 'itm'), ('01', 'Pattern_1', 'Site_p2p1',
'HOME#2', 'device_26', 'itm_1')]
```

FUNC_GET_TEST_OBJ(test_name)

This function gets the test object from the test tree used by the execution engine.

Usage

`FUNC_GET_TEST_OBJ(`*`test_name`*`)`

`test_name`	String; the ID of the test

Returns

- `test_object`: Object
- `None` if the operation fails

Test object properties

- `self.msg`
- `self.commonSetting`
- `self.SMU`
- `self.outputs`
- `self.limit`
- `self.dut`

Test object functions

- `reset()`
- `clear()`

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_TEST_OBJ(ACS_test_id)))

#Output:
<testtree.TEST instance at 0x064CC6E8>

log.logMessage(str(FUNC_GET_TEST_OBJ(ACS_test_id).limit))

#Output:
[{'Target': 0,
'ValidHigh': 9.9999999999999997e+098,
'ConsFail': 0,
'ValidLow': -9.9999999999999997e+098,
'Critical': 0,
'Exit': 'None',
'SpecLow': -9.9999999999999997e+098,
'Report': 1,
'Sigma': 1.0,
'Unit': 'A',
'SpecHigh': 9.9999999999999997e+098}, …]
```

FUNC_GET_SUBSITE_OBJ(subsite_name)

Get the subsite object from the test tree used by the execution engine.

Usage

`FUNC_GET_SUBSITE_OBJ(`*`subsite_name`*`)`

`subsite_name`	String; the ID of the subsite to get

Returns

- `subsite_object`: Object
- `None`: If the operation fails

Subsite object properties

- `self.name`
- `self.id`
- `self.checked`
- `self.DUTList`
- `self.DUTMaps`
- `self.location`
- `self.ssiteList`

- `self.loop`
- `self.site`
- `self.msg`

Subsite object function

`getTest(test_name)`

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_SUBSITE_OBJ(ACS_ssite_id)))

#Output:
<testtree.SUBSITE instance at 0x068CD0A8>

log.logMessage(str(FUNC_GET_SUBSITE_OBJ(ACS_ssite_id).name))

#Output:
subsite
```

FUNC_GET_DUT_OBJ(dut_name)

This function gets the DUT object from the test tree used by the execution engine.

Usage

`FUNC_GET_DUT_OBJ(`*`dut_name`*`)`

dut_name	String

Details

Use the DUT object to get properties and to be able to call functions.

DUT object properties

```
self.info = {
        'devType' : default 'NMOS',
        'checked' : default '1',
        'expand'  : default '0',
        'comment',
        'bitmapFile'     : bitmap file path and name,
        'numberOfSubdev' : default 1,
        'subdevList'     :
                         [
                           [string subdev_ID, [[string padID, [int
                                 SMUID, string padName]]
                                    ...
                                     ]
                                ],
                                ...
                          ]
    }
```

DUT object functions

- `getSMUInfo(subDevNo=-1)`: Get SMU information according to subdevice number
- `reset()`: Reset SMU information to default settings

Returns

- `dut_object`: If operation is successful
- `None` if operation fails

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_DUT_OBJ(ACS_dut_id)))
```

Output:

```
<testtree.DUT instance at 0x06293CD8>
```

FUNC_GET_DUT_NAME(test_name)

This function gets the name of the DUT according to the test name. This is the same value as `ACS_dut_id`.

Usage

`FUNC_GET_DUT_NAME(`*`test_name`*`)`

`test_name`	String

Returns

- `dut_name`: A string if the name of the DUT is returned successfully
- `None` if the operation was not successful

Example

```
import LogManager as log
log.logMessage(ACS_dut_id)
log.logMessage(FUNC_GET_DUT_NAME(ACS_test_id))
```

FUNC_GET_KDF_OBJ(dut_name)

This function gets the kdf object from the test tree used by the execution engine.

Usage

FUNC_GET_KDF_OBJ(*dut_name*)

dut_name	The name of the kdf object

Details

The KDF object contains the following members:

```
head {
    'typ'   : string type,
    'lot'   : string lot ID,
    'prc'   : string process ID,
    'dev'   : string device ID,
    'tst'   : string test program name,
    'sys'   : string tester name,
    'tsn'   : int tester number,
    'opr'   : string operator name,
    'stt'   : string start time,
    'sk1'   : string search key 1,
    'sk2'   : string search key 2,
    'sk3'   : string search key 3,
    'lmt'   : string limit file name,
    'wdf'   : string wafer description file name,
    'com'   : comment
}
wafers   [
    {
        'ID'    : string ID,
        'split' : int split,
        'boat'  : int cassette,
        'slot'  : int slot,
        'sites' : [
                {
                 'ID'    : string ID,
                 'coord' : (int x, int y),
                 'data'  : [
                     [string ID, float scalar | [float arrayitem, ...]]
                         :
                         :
                 ]
            }
            :
            :
        ]
    }
    :
    :
]
```

```
        waferIDs [[string ID, int count], ...]
        siteIDs  [[string ID, int count], ...]
        dataIDs  [[string ID, int count], ...]
        msg      string or None
```

Example

```
import LogManager as log
log.logMessage(str(FUNC_GET_KDF_OBJ()))
```

Output:

```
<KDF.KDF instance at 0x064CB738>
```

Tools for UAP routines

You can use the `import` command to reuse defined modules in another program. See the following topics to learn how to use these modules for your program.

Importing Python modules

You can use one of the following methods to import a module.

`import X`	Imports the module specified by `X`. After you have run this statement, you can use the `X` name to refer to things defined in module `X`.
`import X as Y`	Renames the module `X` as `Y`. If `X` is a long name, you should use a short name for `Y`.
`from X import *`	Imports all public objects from the module `X`. This allows you to use the name to refer to things defined in module `X`. If the `X` itself is not defined, the `X` name is not valid. If the name is already defined, it is replaced by the newer version. If the name in `X` is changed to point to some other object, do not import it because the module will not be recognized.

NOTE

From the `X` import `a, b, c`: This imports `a, b, c` objects from the module `X`. You can now use `a`, `b`, and `c` in your program. It is recommended that you always use the import `X` statement.

Modules in ACS

> **NOTE**
>
> If you want to use these modules, you must import them into a user access point (UAP) routine using the `import` statement. For example, `import LogManager`.

LogManager

The `LogManager` module provides functions to print information to the log window. The following functions print information according to log level.

`logMessage(message)`

`logDebug(message)`

`logInfo(message)`

`logWarning(message)`

`logError(message)`

`logCritical(message,color=False)`: When `color` is `True`, the message is printed in red.

logCritical(message, color=True|False)

When `color` is set to `True`, the message is printed in red.

.klf

The `.klf` module provides functions to operate on a limits file and is used with the global variable `ACS_temp_klf`.

You must construct a `.klf` object before using these functions. Refer to the *Automated Characterization Suite (ACS) Fundamental Reference Manual* (document number ACS-914-01) for more information. The following table contains example `.klf` functions.

Example function	Description
`KLF.KLF(klf)`	Constructs a KLF object. The `.klf` file is loaded into the KLF object.
`.load(klf)`	Loads a KLF file.
`.clear()`	Clears the KLF object; `head` is set to `{}`, `limits` is set to `[]`, and `msg` is set to `None`.
`.save(klf, limits)`	Saves the limit list into a `.klf` file.
`.find(ID)`	Searches `ID` values in the limit list, and returns a reference of the item (if found).
`.remove(ID)`	Removes an item from the limit list according to the `ID` value given.
`.new(ID = '')`	Creates a new limit item with all default values.
`.reset(ID_or_lim)`	Resets a limit item with default values; the ID of the item is reserved.
`.update(ID_or_lim, **attrib)`	Updates a limit item with the new attributes given.
`.insert(index, ID_or_lim)`	Inserts an item at the specified index of the limit list.
`.append(ID_or_lim)`	Appends an item at the end of the limit list.
`.validate(ID_or_lim, autofix = False)`	Validates a limit item.
`.test(ID, value):`	Tests whether a parameter meets the specification; `valid` or `invalid`.

kimisc

The `kimisc` module provides some common operations used in ACS. The following table lists example functions.

Example function	Description
`atoi(str, base = 10)`	Converts a string into an integer using the optional base (default is decimal). The difference between this function and Python standard `string.atoi(x[,base])` is this function allows you to input a string with characters other than decimal digits. The `atoi` function attempts to parse as many characters as it can to build an integer.
`atof(val)`	Converts a string into float. The difference between this function and Python standard `string.atof(x)` is this function does not force you to input a pure float. `atof()` attempts to parse as many characters as it can to build a float.
`isnan(num)`	Whether num is `-1.#IND`.

`getSetting(filename, section, item, openc = '[', closec = ']', asignc = '=', multi_asignc = False, matchFlag = 0)`	This is an operation for a `.ini` file. Extracts a setting from a file. The file has a format similar to:
	`[Section 1]`
	`Item1=Setting 1`
	`Item2=Setting 2`
	`Item3=Setting 3`
	`:`
	`[Section 2]`
	`Item1=Setting 1`
	`Item2=Setting 2`
	`:`
	`[Section 2]`
	`Item1=Setting 1`
	`Item2=Setting 2`
	`:`

`putSetting(filename, section, item, value, openc = '[', closec = ']', asignc = '=')`	This is an operation for a `.ini` file. Corresponding to `getSetting`, this function sets the item value for the section.
`delSetting(filename, section, item, openc = '[', closec = ']', asignc = '=')`	This is an operation for a `.ini` file. This function deletes item of the section.
`avgsdev(data)`	Gets average and standard deviation of data. If the data type is float or int, returns (data,0); if the data type is list and len, (data)=1 returns (data[0],0) if data is float list, return [average,standard deviation]
`copy_tree(src, dest)`	Copies a directory from the source (`src`) to a destination (`dest`).
`getfoldersize(dir)`	Computes the size of the directory in KB.

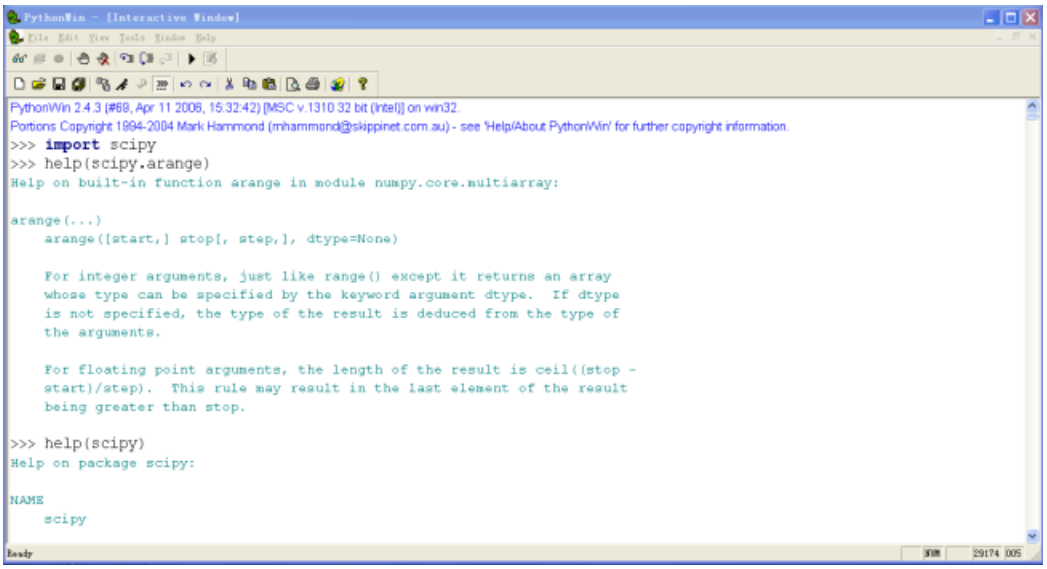
Modules in Python

NOTE

All Python standard and extension modules can be used in user access points (UAPs). If you want to use these modules, you must import them to a UAP routine using the import statement (for example, `import scipy`).

You can use the `help()` function with all Python modules to get the documentation for the module or function in a PythonWin interactive window.

Figure 98: Sample help() description



copy

You can use the deep copy operation (`deepcopy(obj)`) on arbitrary Python objects. It is mostly used with recursive objects (objects that contain another object).

numpy

The `numpy` module includes the following operations.

Operation	Description
`sin()`	Gets the sin value.
`pi`	The constant pi.
`array`	Returns an array from an object with the specified data type.
`sum`	Sums the array over the given axis.

os

The `os.path` module provides many useful common pathname manipulations. For details in a PythonWin window, see `help(os.path)`.

Operation	Description
`os.path.join(a, *p)`	Join two or more path name components, inserting '\' as needed.
`os.mkdir(path)`	Create a directory. If an intermediate directory does not exist, an error is raised.
`os.makedirs(path)`	Super-mkdir. Recursively creates a directory. If a directory does not exist, it creates it.

scipy

This Python module contains the following operations:

- `min`
- `max`
- `median`
- `average`
- `stddev`

shutil

The `shutil` Python module contains the `shutil.copy2(srcname, destname)` operation, which copies data and all state information.

string

The string Python module contains the `string.atof(s)` operation, which returns the floating-point number represented by the string `s`.

time

The time Python module contains the following operations.

Operation	Description
`time.sleep(t)`	Delays execution for a given number of seconds.
`time.strftime (format[, tuple])`	Returns a format string, for example: `time.strftime('%Y.%m.%d %H:%M:%S', time.localtime())`

types

The `types` Python module defines names for all the symbol types recognized in the standard interpreter. For additional details, use `help(types)` in PythonWin after you import the types.

xls

The `xls` Python module uses the .xml format. For additional detail, use `help(xls)` in PythonWin after you import `.xls` file.

Modules in wxpython

You can use the following modules in wxpython.

Module	Description
`wx.MessageDialog()`	Opens a popup message window.
`wx.TextEntryDialog`	Opens a popup text entry dialog box and gets the input value.
`wx.grid`	

.dll modules

You can use the `KIGPIB_KATS` .dll module.

File operation

Refer to the `help(file)` description in PythonWin for more information about file operations. To get information about common path name manipulation routines, refer to `help(operating system.path)`.

You can use the following file operations for user access point (UAP) routines.

Operation	Description
`file(name[, mode[, buffering]])`	Opens a file. The mode can be `r` for reading (default), `w` for writing, or `a` for appending.
`open()`	An alias for `file()`.
`write(str)`	Writes the string `str` to the file.
`read([size])`	Reads at most size bytes and returns as a string.
`close()`	Closes the file.

NOTE

All library and module names are case sensitive.

How to use a UAP

You can use user access point (UAP) routines for many purposes:

- Control the test process
- Calculate parameter values for a site or wafer
- Write data to a file
- Write data to a remote database
- Transfer data to an FTP site

The following topics describe how to use a UAP (Control the test process (on page 6-36) and Write data to a file (on page 6-36)).

Control test process

You can change the normal test process by changing some global variable flags according to a certain condition. For example, you can change `FUNC_SET_KTXE_LOOP_WAFER()` and `FUNC_EXIT_KTXE()`.

You can also halt the execution process by displaying a text input dialog box or message dialog box under some conditions.

There are three example files (`ChooseFileToSave.py`, `OKCancelDialog.py`, and `QueryEntryDlg.py)` in the following directory that show how to call a wxpython dialog box from a UAP routine: `C:\ACS\library\pyLibrary\UAP`.

Write data to a file

There are two example files provided in ACS software that show how to write data to a file from a user access point (UAP) routine. Refer to the code in `ChooseFileToSave.py` or in the following directory for additional details: `C:\ACS\library\pyLibrary\UAP`.

Keithley Instruments
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • 1-800-833-9200 • tek.com/keithley

KEITHLEY
A Tektronix Company

07/2020