

# Automated Characterization Suite (ACS)

## Programmer's Manual

ACS-907-01 Rev. E / December 2014



ACS-907-01

A Greater Measure of Confidence



**ACS**

**Automated Characterization Suite (ACS)**

**Programmer's Manual**

© 2014, Keithley Instruments

Cleveland, Ohio, U.S.A.

All rights reserved.

Any unauthorized reproduction, photocopy, or use of the information herein, in whole or in part, without the prior written approval of Keithley Instruments is strictly prohibited.

All Keithley Instruments product names are trademarks or registered trademarks of Keithley Instruments. Other brand names are trademarks or registered trademarks of their respective holders.

Document number: ACS-907-01 Rev. E / December 2014

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 qualified 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 Instruments 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, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.


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 danger. 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 caution, 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 dangers 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.

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 Instruments. Standard fuses with applicable national safety approvals may be used if the rating and type are the same. 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 Instruments to maintain accuracy and functionality of the product). If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean an instrument, use a damp cloth 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 January 2013.

# Table of Contents

---

<b>Programming overview .....</b>	<b>1-1</b>
ACS programming methods.....	1-1
Creating PTM (or STM) test libraries and modules .....	1-2
<b>LPT Library Reference.....</b>	<b>2-1</b>
TSP LPT library introduction .....	2-1
Series 2600B TSP LPT library commands .....	2-1
Python LPT library.....	2-14
Python LPT introduction .....	2-14
Python LPT functions .....	2-15
ACS LPT library commands .....	2-17
PTM examples .....	2-49
<b>Python Test Module (PTM) Debug Tool.....</b>	<b>3-1</b>
PTM debug tool introduction .....	3-1
PythonWin description .....	3-2
PTM debugging .....	3-4
Debug tool limitations.....	3-20
<b>Series 2600B Library and Python Library .....</b>	<b>4-1</b>
Series 2600B library introduction .....	4-1
Create a library without Script Editor.....	4-3
Create a library using Script Editor.....	4-4
Device library .....	4-4
BJT library overview.....	4-4
MOSFET library overview .....	4-12
Diode library overview.....	4-21
Resistor library overview .....	4-23
WLR library overview .....	4-24
HCI.....	4-25
TDDB_CCS.....	4-30
TDDB_per_pin .....	4-33
NBTI.....	4-36
NBTI_meas .....	4-38
NBTI_on_the_fly .....	4-42
qbd_rmpj.....	4-44
qbd_rmpv .....	4-47
Em_iso_test .....	4-51
Python user library introduction .....	4-53
Configure a capacitor meter library .....	4-54
Configure a switch matrix library .....	4-60
Configure a scope library .....	4-66
Configure a Series 23x library .....	4-67
Configure a Series 3700 system switch DMM library.....	4-74
Configure a Series 2400 SourceMeter instruments library.....	4-75

<b>UAP and Global variable definitions.....</b>	<b>5-1</b>
Overview .....	5-1
Global variables .....	5-2
Global functions .....	5-7
FUNC_SET_GLOBAL_VALUE (name, value) .....	5-8
FUNC_GET_GLOBAL_VALUE (name) .....	5-8
FUNC_SET_USER_GLOBAL (name).....	5-9
FUNC_SET_KTXE_LOOP (value).....	5-9
FUNC_SYS_GLOBAL (name, value).....	5-9
FUNC_EXIT_KTXE () .....	5-9
FUNC_SET_KTXE_SUBSITES (subsite list) .....	5-9
FUNC_GET_KDF_HEADER () .....	5-10
FUNC_SAVE_KDF_HEADER (kdf_file, header).....	5-11
FUNC_SAVE_KDF_WAFERID (kdf_file, wafer_id).....	5-11
FUNC_SAVE_KDF_EOW (kdf_file) .....	5-11
FUNC_SAVE_KDF_SITEID (kdf_file, site_id, site_x, site_y) .....	5-12
FUNC_SAVE_KDF_EOS (kdf_file) .....	5-12
FUNC_SAVE_KDF_DATA (kdf_file, data) .....	5-12
FUNC_GET_KDF_DATA (wafer_id, site_id).....	5-13
FUNC_GET_NEW_KDF_DATA (wafer_id, site_id).....	5-14
FUNC_SITE_COORD_2_ID (coord).....	5-14
FUNC_SITE_ID_2_COORD (site_id).....	5-15
FUNC_GET_WDF_OBJ () .....	5-15
GET_EXEC_TEST_DIC () .....	5-16
FUNC_EXEC_TEST (location).....	5-16
GET_EXEC_TEST_LIST () .....	5-17
FUNC_GET_TEST_OBJ (test_name).....	5-18
FUNC_GET_SUBSITE_OBJ (subsite_name).....	5-19
FUNC_GET_DUT_OBJ (dut_name) .....	5-20
FUNC_GET_DUT_NAME (test_name) .....	5-21
FUNC_GET_KDF_OBJ (dut_name) .....	5-22
Tools for UAP routines .....	5-23
Importing python modules.....	5-23
Modules in ACS .....	5-24
Modules in python .....	5-25
Modules in wxpython.....	5-26
.dll modules .....	5-26
File operation .....	5-27
How to use UAPs .....	5-27
Control test process .....	5-27
Write data to a file .....	5-27

---

## Programming overview

### In this section:

---

ACS programming methods .....	1-1
Creating PTM (or STM) test libraries and modules .....	1-2

## ACS programming methods

The ACS Software allows the user to create and sequence measurements in several ways:

- **Interactive Test Modules (ITM)** - When this method is used, the user interactively defines tests by assigning instrument resources to the various nodes of a DUT and indicates how what each instrument will force and measure by filling in fields in the ITM GUI.
- **Script Test Modules (STM)** - In this method, the user assumes all control of the instrumentation. When the user creates a test, he will create a script that executes on the 2600B-series SMUs or 700B-series switches. These scripts will make calls to the Keithley Instruments Linear Parametric Test Library (LPTLib), Test Script Processor Library (TSP), or Lua programming language statements. This method of programming provides the user with the ability to generate test code that can run at the fastest possible speed since these functions will literally execute on the instruments themselves.
- **Python Test Modules (PTM)** - Similar to the STM case, the user creates PTM modules using calls to LPTLib. Unlike the STM case, PTM modules use the Python programming language and do not execute completely on the instrumentation. PTM modules are supported by all instruments that ACS supports (not just 2600B-series or 700-series) instrumentation. An added feature of creating PTM tests, is the ability to create test modules with customer Graphical User Interfaces.

### NOTE

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

- **C Test Modules (CTM)** - These tests are created using the C Programming Language. This method of test creation is most useful when interfacing with external DLL libraries or when ACS is running directly on a Keithley 4200-SCS. There is no access to LPTLib when this method is used. This method of programming also requires the optional Microsoft Visual Studio software. 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 covers how to use the Keithley LPTLib instrument functions, as well as introducing you on how to create PTM and STM modules. This manual will not cover creating CTM and ITM tests, nor will it show how to program using the Python or Lua programming languages. There are several excellent references on Python and Lua programming:

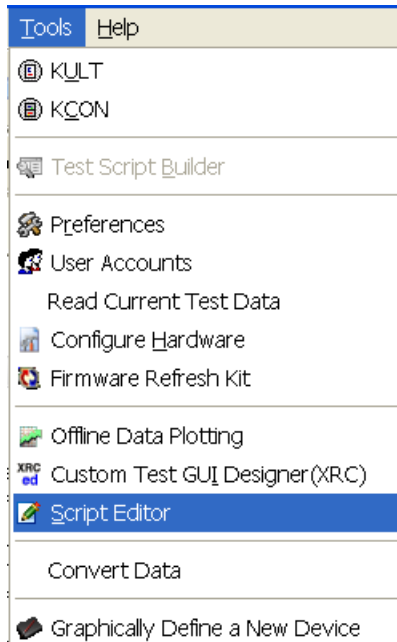
- [www.lua.org](http://www.lua.org)

- Programming in Lua, 2nd Edition, Roberto Ierusalimsky, Lua.org (publisher)
- [www.python.org](http://www.python.org)
- [www.linuxjournal.com/article/3946](http://www.linuxjournal.com/article/3946)
- Programming Python, 4th Edition, Mark Lutz, O'Reilly Media (publisher)

## Creating PTM (or STM) test libraries and modules

To create either a python (PTM) test library or script (STM) test library, the ACS Script Editor is used. To launch the ACS Script Editor, select Script Editor from the ACS Tools menu (see next Figure):

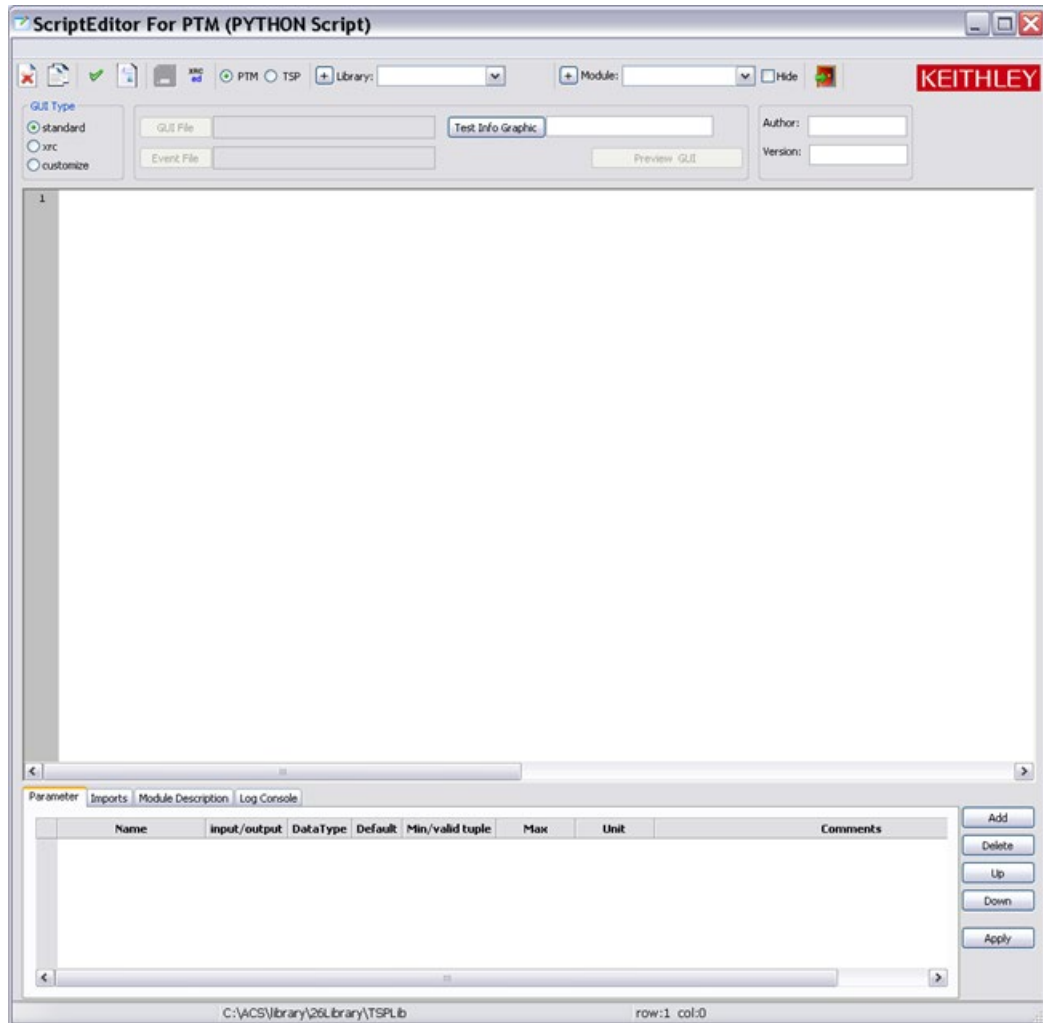
**Figure 1: Script Editor in Tools menu**





The ACS Script Editor will start in a separate window (see next Figure):

**Figure 2: Script Editor for PTM**



The first step in creating a test library is to determine which type of library you want to create. The ACS Script Editor supports either PTM test libraries or TSP test libraries. Select the type of test library you need by clicking on the radio button next to the library type (see next Figure):

**Figure 3: Test library types**



In this first example, a PTM test library will be created. Now that the library type has been made, you must now either create a new library or add to an existing library. In this example, a new library will be created. To do this, click the + (add Library) symbol icon on the script editor toolbar (see next Figure):

## NOTE

To create a STM Library, you will follow a similar series of steps, however, the only differences are that you select the TSP radio button in the ACS Script Editor, use the Lua programming language, and add and import a STM module to your test project.

**Figure 4: Add Library icon on toolbar**



A new library creation dialog box will open. There are several features in the dialog box. You can create a new library by copying from an existing library or create a new library. In this example, a completely new library will be created. First, type in a name for the new library in the New Library field. You can also specify the name of the first new module in the new library at this time. For this example, the new library is named eTest and the first test module named resv (see next Figure):

**Figure 5: Create a new library dialog box**



Click OK after you have entered a name for the New Library and New Module. At this point, the Script Editor indicates the name of the newly created library and first test module in the library name and module name fields:

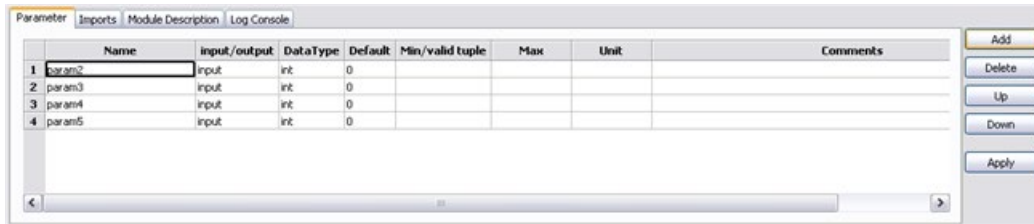
**Figure 6: Library and Module names**



For this example, a module to measure resistance will be created. The resistance module will have three input values: high pin, low pin, and voltage force value. The module will also return the calculated resistance. The inputs will be named hpin, lpin, and vforce. The output will be named resistance.

To create the input and output values, select the Parameter tab at the bottom of the Script Editor window and select the Add function four times, one time for each input or output parameter (see next Figure):

**Figure 7: Script Editor Parameter tab**



Enter the names of the parameters as indicated in the Figure. Since hpin, lpin, and vforce are input parameters, click on the input/output field next to each parameter name and select input. For the output field resistance, select output for this field. Pins are Integer values. Select the Data Type as indicated for each input parameter. Output parameters are special and their data type cannot be selected. Select Apply when done (see next Figure):

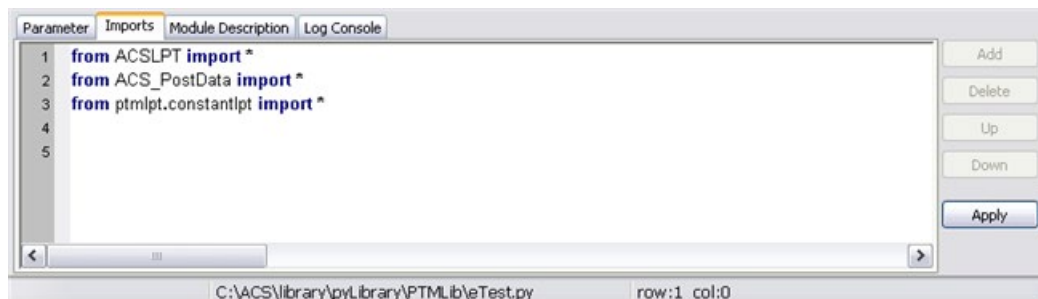
**Figure 8: Names of parameters**

	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			

Once you select Apply, a python function declaration is automatically created in the edit area of the Script Editor. Now it is time to create the actual test code.

Next, each python module must include several imports that enable the test module to locate and use LPTLib and the ACS data handling functions. To do this, select the Imports tab at the bottom of the ACS Script Editor and type the following text as indicated in the next Figure:

**Figure 9: Script Editor Imports tab**



Select Apply when done.

For the resv module, you will use the switch matrix to connect the pins to the SMUs, force voltage using the SMU, measure current using the SMU, calculate the resistance using python, and return the data to ACS. Type the following text as indicated in the next Figure. Since python is a positional language, the editor will automatically indent each statement one tab stop (see next Figure).

**Figure 10: Script Editor resv module information**

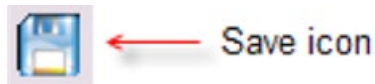
```

1 def resv(hpin=0, lpin=0, vforce=0, resistance='resistance'):
2     conpin(SMU1, hpin)      #Clear the instruments and connect the high terminal
3     addcon(lpin, GNDU)     #Connect the low terminal to ground
4     forcev(SMU1, vforce)   #Force voltage
5     temp_current = measi(SMU1) #Measure current
6     devint()              #Clear the instruments and connections
7     temp_resistance = vforce/temp_current #Calculate resistance
8     ACSPostDataDouble("resistance", temp_resistance) #Return the data to the sheet
9

```

At this point, the module is complete. Make sure you save your work for the current module and library before closing the project by clicking the Save icon on the ACS Script Editor toolbar (see next Figure):

**Figure 11: Script Editor Save icon**



## NOTE

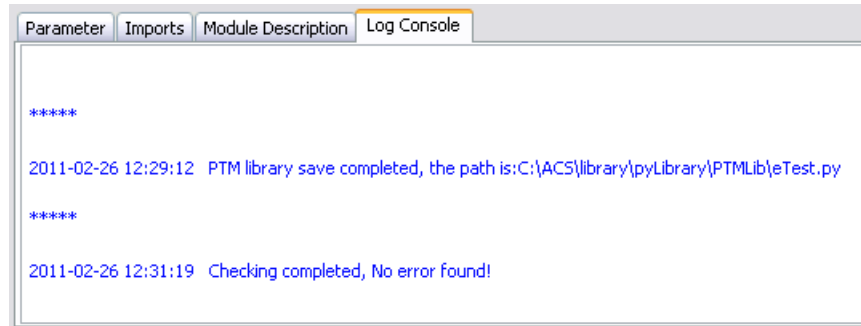
It is recommended that you check your new module to make sure that there are no syntax errors. To scan for syntax errors, click the check function in the ACS Script Editor toolbar (see next Figure):

**Figure 12: Script Editor check function**



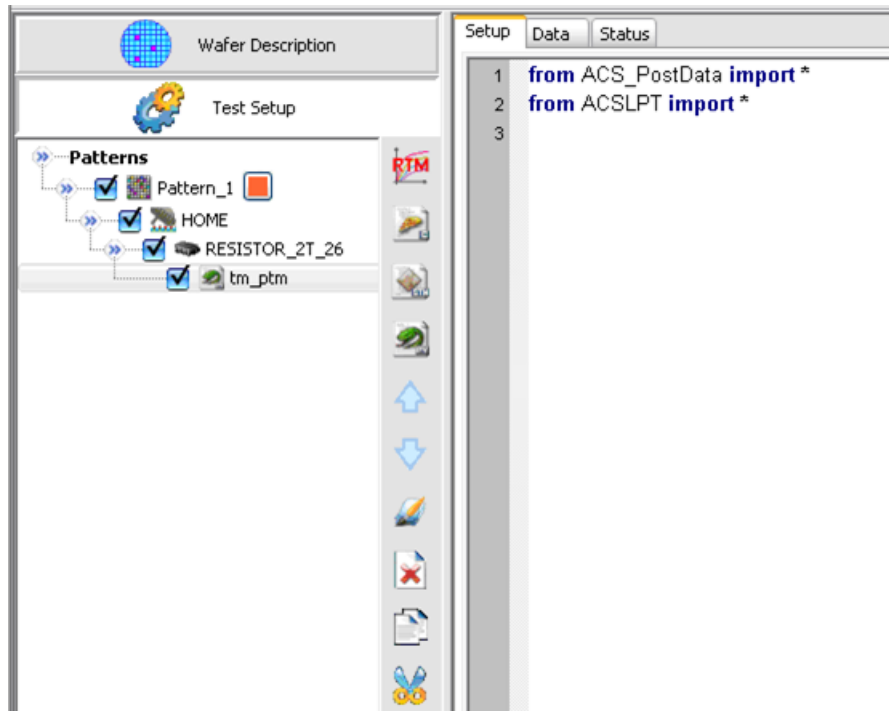
If there are no errors, a message will open similar to the next Figure (in the Script Editor Log Console tab):

**Figure 13: Script Editor Log Console tab**



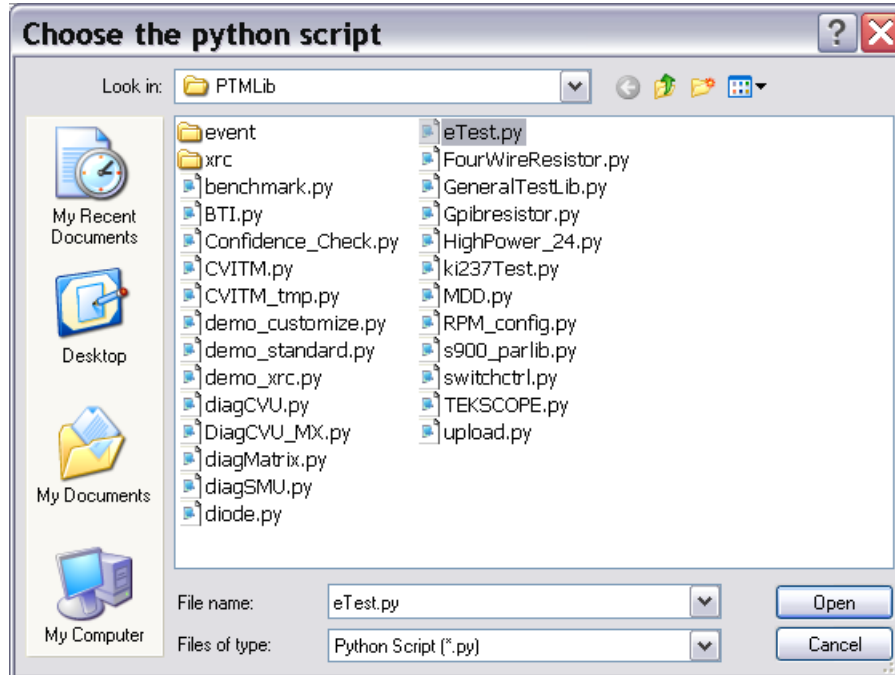
The newly created library and module are now ready to use. To use the module, add a new PTM to the desired ACS project (see next Figure):

**Figure 14: ACS PTM project**



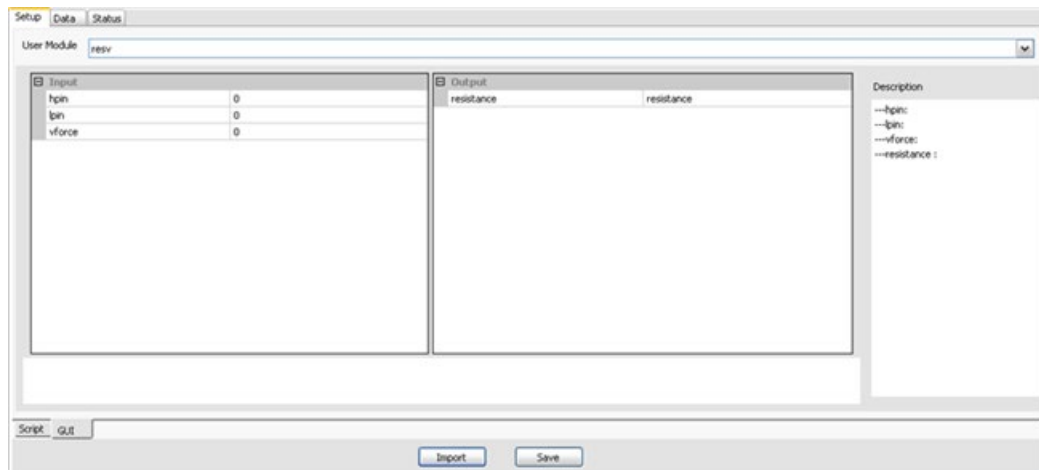
After the new PTM is added, select the Import button at the bottom of the ACS Setup tab. A dialog box opens and you must choose the desired library (in this case eTest.py) from the list. Select the Open function (see next Figure):

**Figure 15: ACS Choose the python script**



After you open the python script, the ACS Setup tab displays a GUI input form for the newly created test module. To use the module, input the pin numbers and force voltage to use and click the Save icon in ACS.

**Figure 16: ACS Setup tab GUI**



Now that you have added the module to the test project, it is ready for use. If you added more than one module to the library, you can select the desired module by clicking on the down arrow in the User Module field and select a different module:

To create a STM Library, you will follow a similar series of steps, however, the only differences are that you select the TSP radio button in the ACS Script Editor, use the Lua programming language, and add and import a STM module to your test project.

---

## LPT Library Reference

### In this section:

---

TSP LPT library introduction .....	2-1
Python LPT library.....	2-14

## TSP LPT library introduction

The Keithley Instruments Linear Parametric Test Library (LPTLib) is a high-speed data acquisition and instrument control software library. It is the programmer's lowest level of command interface to the system's instrumentation.

The Keithley Instruments Automated Characterization Suite incorporates two LPT libraries. One of the LPT libraries contains commands that are compatible with the Keithley Instruments Series 2600B System SourceMeter®.

This is the ACS TSP LPT Library. Most of the commands in the ACS TSP LPT Library contain the same format as those in the Model 4200-SCS library.

The ACS TSP LPT library is built with TSP builder and is programmed with Lua language. They can be used in STM. For more information about STM, refer to Configuring a Script Test Module (STM).

### NOTE

When the Series 2600B System SourceMeter® instruments are referenced, it also includes the Series 2600A System SourceMeter instruments, since these two series of instruments are fully interchangeable. However, the following instruments are not supported in ACS Basic: Model 2604B, Model 2614B, and Model 2634B.

### NOTE

The Keithley Instruments Series 2600B System SourceMeter includes its own Instrument Control Library (ICL). Refer to the Series 2600B Reference Manual for detailed information.

## Series 2600B TSP LPT library commands

### NOTE

The Series 2600B LPT commands are listed in alphabetical order.



## avgi/avgv

---

**Purpose:** Performs a series of measurements and averages the results.

**Format:**

```
avgi(SMUX, Itable, step_num, step_time) X = SMU number(1,2,3,...)
avgv(SMUX, Vtable, step_num, step_time) X = SMU number(1,2,3,...)
```

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 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 0.0001s (npic must be set as 0.001).

## clrscn

---

**Purpose:** Clears the measurement scan tables associated with a sweep.

**Format:**

```
clrscn()
```

## crtbf

---

**Purpose:** Creates a buffer for a specified SMU to store its measurements.

**Format:**

```
buff_name = crtbf(SMUX, buff_cap, append_flag, timestamp_flag) X = SMU
number(1,2,3,...)
```

buff_name	The name of the buffer to be created.
buff_cap	The capacity of the buffer to be created.
append_flag	Use KI_EBAP to enable buffer append mode, KI_DBAP to disable buffer append mode.
timestamp_flag	Use KI_EBTS to enable collecting buffer timestamps, KI_DBTS to disable collecting buffer timestamps.

## delay/rdelay

---

**Purpose:** Provides user-programmable delay within a test sequence. The units are in seconds.

**Format:**

```
delay(second)
rdelay(second)
```

## devclr

---

**Purpose:** Sets all sources to zero.

**Format:**

```
devclr()
```

---

**devint**

---

Purpose: Resets all instruments.

Format:

```
devint()
```

---

**enable**

---

Purpose: Provides realtime measurements of voltage, current, conductance, and capacitance.

Format:

```
enable(ntimer[Y]) Y = Timer number(1,2,3,...)
```

---

**disable**

---

Purpose: Stops the timer and sets the time value to zero. Timer reading is also stopped.

Format:

```
disable(ntimer[Y]) Y = Timer number(1,2,3,...)
```

---

**forceclr**

---

Purpose: Turns the source output off on the specified SMU.

Format:

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

---

**forcei/forcev**

---

Purpose: Programs a sourcing instrument to generate a voltage or current at a specific level.

Format:

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

---

**intgi/intgv**

---

Purpose: Performs voltage or current measurements averaged over a user-defined period (usually one AC-line cycle). This averaging is done in the hardware by integration of the analog measurement signal over a specified time period. The integration is automatically corrected for 50 or 60Hz power mains.

Format:

```
intgi(SMUX, Itable) X = SMU number(1,2,3,...)  
intgv(SMUX, Vtable) X = SMU number(1,2,3,...)
```

Itable The table created by You; the measured value is saved into Itable[1].

Vtable The table created by You, the measured value is saved into Vtable[1].

### **iolv/iolv/ioliv**

---

Purpose: Measure current, voltage, or current and voltage using overlap mode. The integration time is set by `setmode()`, and the measure count is set by `setcount()`. The only difference between this function and `msoli()` is the integration time (`msoli()` uses fixed 0.001 nplc).

Format:

```
iolv(SMUX, i_buff_name) X = SMU number(1,2,3,...)
iolv(SMUX, v_buff_name) X = SMU number(1,2,3,...)
ioliv(SMUX, i_buff_name, v_buff_name) X = SMU number(1,2,3,...)
```

`i_buff_name`      The buffer to store current measurements. The buffer must be created by `crtbf()`, and must be created for the same SMU.

`v_buff_name`      The buffer to store voltage measurements. The buffer must be created by `crtbf()`, and must be created for the same SMU.

### **limiti/limitv**

---

Purpose: Allows the programmer to specify a current or voltage limit other than the instrument's default limit.

Format:

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

### **lorangei/lorangev**

---

Purpose: Defines the bottom auto range limit for current or voltage measurements.

Format:

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

### **measi/measv/meast**

---

Purpose: Allows the measurement of voltage, current, or time.

Format:

```
measi(SMUX, Itable) X = SMU number(1,2,3,...)
measv(SMUX, Vtable) X = SMU number(1,2,3,...)
meast(ntimer[Y], Ttable) Y = Timer number(1,2,3,...)
```

`Itable`      The table created by You. The measured current value is saved into `Itable[1]`.

`Ttable`      The table created by You. The measured time value is saved into `Ttable[1]`.

`Vtable`      The table created by You. The measured voltage value is saved into `Vtable[1]`.

## **moli/molv/moliv**

---

Purpose: Measures current (moli), voltage (molv), or current/voltage using overlap mode (moliv) using a fixed 0.001 nplc.

Format:

```
moli(SMUX, i_buff_name) X = SMU number(1,2,3,...)
molv(SMUX, v_buff_name) X = SMU number(1,2,3,...)
moliv(SMUX, i_buff_name, v_buff_name) X = SMU number(1,2,3,...)
```

**i\_buff\_name**        The buffer to store current measurements. The buffer must be created by `crtbf()`, and must be created for the        same SMU.

**v\_buff\_name**        The buffer to store voltage measurements. The buffer must be created by `crtbf()`, and must be created for the        same SMU.

## **postscript**

---

Purpose: Prints a list of scripts that are currently stored in the master of the Series 2600B instruments, according to the location parameter.

Format:

```
postscript(location)
location = 0: volatile memory
location = 1: nonvolatile memory
```

Default location value: 1

## **postbuffer**

---

Purpose: Prints buffered data to a GPIB output buffer in binary format. ACS software can only recognize buffered data printed by the `postbuffer` function.

Format:

```
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.

**startin\_dex**        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 will automatically calculate the average result of each `avg_num` value. If this parameter is not given by you, the system will give a default value of 1 (print every value point).

---

## postbuftime

---

Purpose: Prints timestamps of buffered data in binary format. ACS software can only recognize buffered timestamp data printed by the postbuftime function.

Format:

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

name	A string that presents the values in the script, defined by 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 will automatically calculate the average result of each avg_num value. note for the same buffer, always use the same avg_num with the one in postbuffer(), or the timestamps' number will not match with the values' number. If this parameter is not given by You, the system will give a default value of 1 (print every value point).

---

## postdata

---

Purpose: Prints a single value. ACS software only recognizes single values printed by the post data function.

Format:

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

name	A string that represents the value in the script, defined by the script writer.
value	The value to print (for example, it could be an execution like "node[2].smua.measure.i()", or "measi(SMU1)").

---

## posterror

---

Purpose: Prints all errors in the error queue separately. This function was designed for the DATA Class (Engine) of ACS.

Format:

```
posterror()
```

---

## postglobal

---

Purpose: Prints all global variables in the realtime memory of the Series 2600B.

Format:

```
postglobal()
```

---

## postsmuinfo

---

Purpose: Prints information for all SMUs.

Format:

```
postsmuinfo()
```

## posttable

---

Purpose: Prints table data. Each item in the table must be a numeric value.

Format:

```
posttable("name", table_name)
```

## rangei/rangev

---

Purpose: Clicks current/voltage measurement range and prevents the selected instrument from autoranging. By clicking a range, the time required for autoranging is eliminated.

Format:

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

## savgi/savgv

---

Purpose: Performs an averaging current or voltage measurement for every point in a sweep.

Format:

```
savgi(smu_num, Itable, step_num, step_time) X = SMU number(1,2,3,...)
savgv(smu_num, Vtable, step_num, step_time) X = SMU number(1,2,3,...)
```

Itable	The table created by user; the measured value is saved into Itable[1]
Vtable	The table created by user; the measured 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.

## scnmeas

---

Purpose: To perform a single measurement on multiple instruments at the same time.

Format:

```
scnmeas()
```

Remarks: 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 prior to calling scnmeas. smeasX, sintgX, or savgX must be used to set up result arrays just as is done for a sweep call. Each call to scanmeas will add one element to the end of each array. Calls to scnmeas may be mixed with calls to sweepX and all results will be appended to the result arrays in the same way multiple sweepX calls behave.

## setauto

---

Purpose: Sets SMU measurement auto range.

Format:

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

## setcount

Purpose: Sets the number of measurements performed when a measurement is requested.

- 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.

Format:

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

## setitv

Purpose: Sets the interval between multiple measurements. The unit of value is seconds.

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 will be made as fast as possible.
- The reset function sets the measure interval to 0.

Format:

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

## setmode

Purpose: Set instrument-specific operating mode parameters. Modifies instruments' specific operating characteristics (see next table).

Format:

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

Setmode parameters			
Parameters			Comment
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 / KI_INTEGRATE	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_OFFMODE	KI_OFF_NORM / KI_OFF_ZERO / KI_OFF_OPEN	Set source output-off mode. KI_OFF_NORM: Outputs 0 V 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_SENSE	KI_SENSE_LOCA / KI_SENSE_REMO / KI_SENSE_CALA	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.

---

### **sintgi/sintgv**

---

Purpose: Performs an integrated current or voltage measurement for every point in a sweep.

Format:

```
sintgi(SMUX, Itable) X = SMU number(1,2,3,...)
sintgv(SMUX, Vtable) X = SMU number(1,2,3,...)
```

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].

---

### **slorangei/slorangev**

---

Purpose: Defines the bottom autorange limit for current or voltage source.

Format:

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

---

### **smeasi/smeasv/smeast**

---

Purpose: Allows a number of current/voltage/time measurements to be made by a specified instrument during a sweepX function. The results of the measurements are stored in the defined array.

Format:

```
smeasi(SMUX, Itable) X = SMU number(1,2,3,...)
smeasv(SMUX, Vtable) X = SMU number(1,2,3,...)
smeast(ntimer[Y], Ttable) Y = Timer number(1,2,3,...)
```

Itable The table created by You; the measured current value is appended into Itable.

Vtable The table created by You; the measured voltage value is appended into Vtable.

Ttable The table created by You; the measured time value is appended into Ttable.

---

### **srangei/srangev**

---

Purpose: Clicks the current/voltage source range and prevents the selected instrument from auto-ranging. By clicking a range, the time required for auto-ranging is eliminated.

Format:

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

---

### **ssetauto**

---

Purpose: Sets SMU source to auto range.

Format:

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



---

## sweepi/sweepv

---

Purpose: Generates a ramp consisting of ascending or descending currents or voltages. The sweep consists of a sequence of steps, each with a user-specified duration.

Format:

```
sweepi(SMUX, start, end, step_number, delay_time) X = SMU number(1,2,3,...)  
sweepv(SMUX, start, end, step_number, delay_time) X = SMU number(1,2,3,...)
```

start	The initial voltage or current level output from the sourcing instrument 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_num	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.

---

## sysinit

---

Purpose: Sets nplc to 0.001, measure count to 1; effects every SMU in the system. Clears the error queues and resets all registers.

Format:

```
sysinit()
```

---

## sysquery

---

Purpose: Queries every node and every SMU in the system and gives every SMU a unique name, for instance, SMUX. Displays node number and SMU number on every Series 2600B's screen. Sets the integration nplc to 1 and average mode to KI\_MEASX on every SMU in the system.

Format:

```
sysquery()
```

## 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)

```

local v_value = {}
local i_value = {}
local error = {}
if sensemode ~= 0 and sensemode ~= 1 then
table.insert(error,-10100)
posttable("error",error)
return
end
if testmode ~= 0 and testmode ~= 1 then
table.insert(error,-10100)
posttable("error",error)
return
end
setmode(RSMU1, KI_INTGPLC, myNPLC)           --set RSMU1's NPLC
setmode(RSMU1, KI_SENSE, sensemode)         --set RSMU1 in sensemode
if RSMU2 ~= KI_GND then
setmode(RSMU2, KI_SENSE, sensemode)
limiti(RSMU2, 1)                            --set RSMU2 current limit
end
if testmode == 0 then --if
limiti(RSMU1, myLIMIT)                      --set RSMU1 current limit
forcev(RSMU1, forcevalue)                   --force RSMU1 voltage source value
elseif testmode == 1 then
limitv(RSMU1, myLIMIT)                     --set RSMU1 voltage limit
forcei(RSMU1, forcevalue)                  --force RSMU1 current source value
end
if RSMU2 ~= KI_GND then--if
forcev(RSMU2, 0)                           - -force RSMU2 voltage source value
end
--if
delay(testdelay)                           --set delay time before measure
intgv(RSMU1, v_value)                       --measure RSMU1 voltage
intgi(RSMU1, i_value)                       --measure RSMU1 current
Rvalue[1] = v_value[1]/i_value[1]
posttable("Rvalue", Rvalue)
table.insert(error, 0)
posttable("error",error)
devint()                                     --reset all instruments after test
end
--function
--INPUT--
Local Sensemode = 0
local testmode = 1
local RSMU1 = SMU1
local RSMU2 = KI_GND
local forcevalue = 1e-3
local myLIMIT = 20
local myNPLC = 1
local testdelay = 0.01
local Rvalue = {}

```

```
R_single(sensemode, testmode, RSMU1, RSMU2, forcevalue, myLIMIT, myNPLC, testdelay,
        Rvalue)
---End of Input---
```

## 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)

```
local vg
local i
local Vg_inc
local id_t1={}
local dummy={}
setmode(DSMU, KI_INTGPLC, myNPLC)      --set the NPLC of DSMU
limiti(GSMU,Gcompliancei)              --set current compliance to GSMU
limiti(DSMU,Dcompliancei)              --set current compliance to DSMU
setauto(DSMU)                           --set DSMU measure range to auto
if SSMU~=KI_GND then
limiti(SSMU,Scompliancei)              --set current compliance to SSMU
forcev(SSMU,VSS)                       --apply SSMU voltage source
end
if BSMU~=KI_GND then --if
limiti(BSMU,Bcompliancei)              --set current compliance to BSMU
forcev(BSMU,VBULK)                     --apply BSMU voltage source
end
--if
forcev(DSMU,VD)                         --apply DSMU voltage source
forcev(GSMU,Vg_start)                   --apply GSMU voltage source
delay(holdtime)                          --set time delay before measure
intgi(DSMU,dummy)                       --perform current measure on DSMU
forcev(DSMU,VD)                         --apply DSMU voltage source
timer.reset()
for i=1,Vg_points do
vg=Vg_start+(i-1)*Vg_inc
forcev(GSMU,vg)                         --apply GSMU voltage source
table.insert(Vg,vg)
delay(sweepdelay)                       --set time interval between every point
intgi(DSMU,id_t1)                       --perform current measure on DSMU
table.insert(Id,id_t1[1])
table.insert(time,timer.measure.t())
end
--for
table.insert(error,0)
posttable("error",error)
posttable("time",time)
posttable("Vg",Vg)
posttable("Id",Id)
devint()
end
--function
```

```
-----  
--CALL--  
local DSMU=SMU2  
local GSMU=SMU1  
local SSMU=KI_GND  
local BSMU=KI_GND  
local Vg_start=0  
local Vg_stop=2  
local Vg_points=21  
local Dcompliancei=0.1  
local Gcompliancei=0.1  
local Scompliancei=0.1  
local Bcompliancei=0.1  
local VD=1  
local VBULK=0  
local VSS=0  
local myNPLC=1  
local holdtime=0.01  
local sweepdelay=0.001  
local error={}  
local time={}  
local Id={}  
local Vg={}  
---End of Input---
```

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)

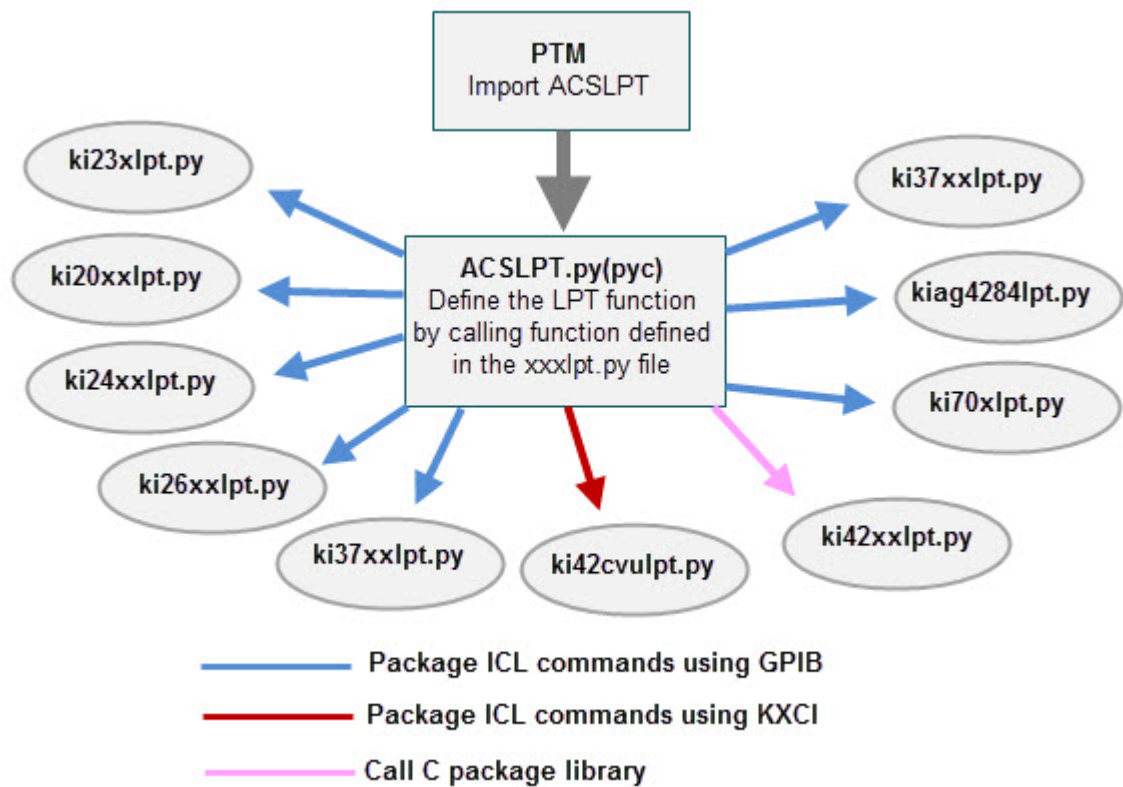
# Python LPT library

## Python LPT introduction

For PTM (Python Test Module), ACS includes another special LPT library: ACSLPT. The ACSLPT has functions that let you configure one or multiple instrumentation to perform parametric tests.

The commands in ACSLPT can be used to configure some general instruments. For example, you can import ACSLPT and PTM can control these instruments: 23x series, Series 2400 SourceMeter instruments, Series 2600B SourceMeter instruments, Series 3700 System Switch, 4200CVU, 4200/4210SMU, matrix 70x series, Model 2010 Multimeter, and LCR 4284/4980 capacitance meter. For more information, you can refer to Configuring a Python Language Test Module (PTM).

Figure 17: ACS LPT call flow



## NOTE

In the following table, you will learn how the CTM modules and the ACS software function and interact (see next Table).

<b>ACS software compatibility</b>		
<b>ACS installed on:</b>	<b>Interface:</b>	<b>Compatible library:</b>
Model 4200-SCS	Normal (non-KXCI)	CTM functions
Model 4200-SCS	KXCI and Ethernet cable	Ki42cvulpt commands
PC	KXCI and Ethernet cable	Ki42cvulpt commands

## Python LPT functions

In the following tables, function calls are grouped by different instruments. The details on functions for the SMUs and general operations are listed alphabetically (see next Tables).

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

<b>LPT functions</b>		
<b>Model 2010 Multimeter LPT functions</b>		
devclr	devint	avgv
intgv	measv	rangev
setauto	setmode	getstatus

## NOTE

The `lorangei` and `lorangev` functions for the 23x are equal to auto range, no matter the value of the parameter setting.

<b>LPT functions</b>			
<b>Series 2400 SourceMeter instruments LPT functions</b>			
abort	delay	devclr	devint
forcei	forcev	intgi	intgv
limiti	limitv	measi	measv
rangi	rangev	setauto	setmode
srangei	srangev	sweepi	sweepv

LPT functions				
Series 2600B SourceMeter instruments LPT functions				
abort	avgi	avgv	devclr	devint
forcei	forcev	intgi	intgv	limiti
limitv	lorangei	lorangev	measi	measv
rangi	rangev	setauto	setmode	srangei
srangev				

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

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

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

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

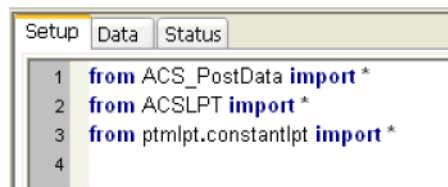
LPT functions			
Model 4284 LCR Meter LPT functions			
devclr	devint	forcev	getstatus
measz	rangei	setauto	setfreq
setlevel	setmode		

## ACS LPT library commands

### NOTE

Before using the ACSLPT commands, you need to import ACSLPT and ptmplt.constantlpt to the header lines of a PTM (see next Figure). The ACSLPT commands are listed in alphabetical order.

Figure 18: Import ACSLPT



### abort

Purpose: To abort the current source-delay-measure process. It is recommended that you call the abort function node in the user access point (UAP) only.

### NOTE

The abort function is only valid with the Series 2600B and Series 2400 instruments (SMUs).

Format:

```
abort (*args)
```

Example:

```
abort (SMU1)
```



## addcon

---

Purpose: Add terminal-pin connections.

Format:

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

`instMTRX` The matrix name in the hardware configuration, it's optional.

`ter` The list of terminals to be connected.

`pin` The list of pins to be connect.

`more_pin` Indicates more pins to connect.

Remarks: Terminal and pin lists must have the same number of items. Terminals and pins will be matched according to the sequence. If the numbers in the terminal and pin lists are not the same, the connection will be performed according to the shortest list.

Normally, `addcon` supports the "ROW\_COLUMN" mode of matrix. When the matrix is set to "INSTRUMENT\_CARD" mode, a row will be assigned automatically to connect the terminal and the pin.

For more information about the how to set the "INSTRUMENT\_CARD" mode and "ROW\_COLUMN" mode, refer to the ACS Reference manual Hardware configurations section.

Example:

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

## addpth

Purpose: Add terminal-pin connections by path.

Format:

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

*instMTRX* The matrix name in hardware configuration, it's optional.

*ter* The list of terminals to be connected.

*pin* The list of pins to be connect.

*row* The row used to connect terminals and pins

Example: 70X

```
addpth(MTRX1, SMU1, 1, 'A')
addpth(SMU1, 1, 'A')
addpth(SMU1H, 1, 'A')
addpth(SMU1L, 1, 'A')
addpth([SMU1, SMU2], [1, 2], 'A')
addpth(MTRX1, [SMU1, SMU2], [], 'A')
addpth([], [1, 2], 'A')
```

Series 3700 System Switch

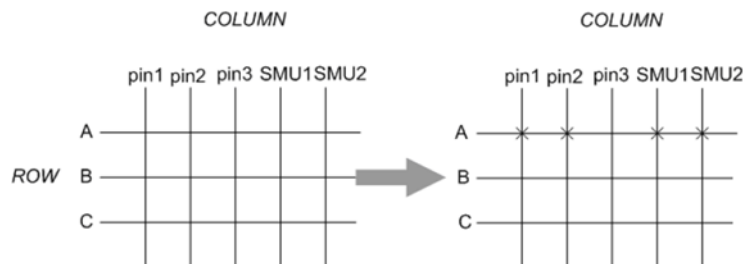
```
addpth(MTRX1, SMU1, 1, '1')
addpth(SMU1, 1, '1')
addpth(SMU1H, 1, '1')
addpth(SMU1L, 1, '1')
addpth([SMU1, SMU2], [1, 2], '1')
addpth(MTRX1, [SMU1, SMU2], [], '1')
addpth([], [1, 2], '1')
```

Remarks: All terminals and pins will be connected together in the row. One command can not connect paths in multiple matrices.

You can also connect only terminals or only pins with this function. But when connecting only terminals, *instMTRX* is required, otherwise the function will not know which instrument to send the command to.

For more information about the how to set the "INSTRUMENT\_CARD" mode and "ROW\_COLUMN" mode, refer to Hardware Configuration.

Figure 19: addpth library command



## avgi/avgv

---

**Purpose:** Performs a series of measurements and averages the results.

**Format:**

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

**iStepNo** The number of steps averaged in the measurement. This number ranges from 1 to 160,000 (for 42xx is 32767).

**dStepTime** The interval in seconds between each measurement. Minimum practical time is approximately 0.0001s (nplc must be set as 0.001, for Model 4200 set as 2.5us).

**Example:**

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

## checkparam

---

**Purpose:** Check the hardware limits parameter according to hwlimits file. Only applies to dc range and limit check.

**Format:**

```
checkparam(unitname, **kwargs)
```

**unitname:** The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**\*\*kwargs:** A dictionary of arbitrary keyword arguments supplied using callback. The names are defined in C:\S4200\sys\kcon\hwlimits.ini ["dc\_srange\_v", "dc\_srange\_i", "dc\_range\_v", "dc\_range\_i", "dc\_lmt\_v", "dc\_lmt\_i"].

**Example usage:**

```
dc_range_v=10, dc_i_lmt=0.1
```

**return value:** dictionary/number

**dc\_range\_v** (INVAL\_PARAM,correct\_range) / (OK, the lowest range if input value less than it) / (OK, input\_range)

**dc\_lmt\_i** (ERR\_CHECKPARAM, input\_range) / (INVAL\_PARAM, correct\_lmt) / (OK, input\_range)

**INVAL\_INST\_ID** invalid instrument ID

**ERR\_CHECKPARAM** An error will be reported if check limits and no source range in input dict.

**Example:**

```
heckparam(SMU1,dc_lmti_i=1, dc_srange_v = 10)
```

## clratrset

---

**Purpose:** Clear current instrument setting saving in memory.

**Format:**

```
clratrset( *args)
```

**\*args:** A tuple of arbitrary positional arguments supplied using the callback\_args option attribute.

**Example:**

```
Clratrset(SMU1, SMU2)
```

## clrcon

---

Purpose: Clear all connections of all the matrixs or specified matrixs.

Format:

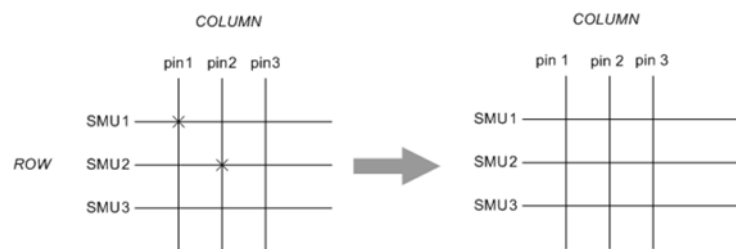
```
clrcon(unitname)
```

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf, such as MTRX1.

Example:

```
clrcon( ) ## Clear all matrixs' connections in the hardware configuration.
clrcon(MTRX1) ## Clear only matrix1's connections.
```

**Figure 20: clrcon library command**



## clrscn

---

Purpose: Clears the measurement scan tables associated with a sweep, only used in the Model 4200-SCS.

Format:

```
clrscn(*args)
```

\*args: A tuple of arbitrary positional arguments supplied using the `callback_args` option attribute.

Example:

```
clrscn(), clrscn(SMU1,SMU2,CVU1)
```

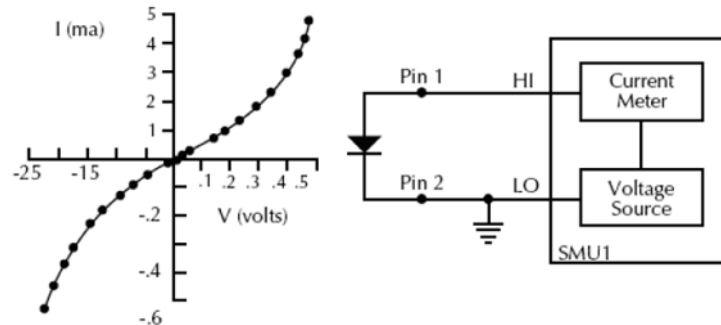
## clrtrg

Purpose: Clears the user-selected voltage or current level used to set trigger points. This permits the use of trigXI or trigXg more than once with different levels within a single test sequence. Only used in Model 4200-SCS.

Format:

```
clrtrg(*args)
```

Figure 21: clrtrg library command



```
conpin(SMU1, 1, 0)
conpin(GND, SMU1L, 2, 0)
trigil(SMU1, 5.0e-3)# Increase ramp to I = 5mA.
smeasi(SMU1, forcur')# Measure forward
sweepv(SMU1, 0.0, 0.5, 10, 5.0e-3)# Output 0 to 0.5V in 10 steps, each 5ms
duration. clrtrg() # Clear 5mA trigger point.
clrscn() # Clear sweepv
trigil(SMU1, -0.5e-3)# Decrease ramp to I = -0.5mA.
cur=smeasi('SMU1')# Measure reverse
sweepv(SMU1, 0.0, -30.0, 10, 5.00e-3)
```

## conpin

---

Purpose: Clear old connections and adds new terminal-pin connections

Format:

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

`instMTRX` The matrix name in hardware configuration, it's optional.

`ter` The list of terminals to be connected.

`pin` The list of pins to be connect.

`more_pin` Indicates more pins to be connect.

Remarks: Normally `conpin()` supports "ROW\_COLUMN" mode of matrix. When matrix is set to be "INSTRUMENT\_CARD" mode, rows will be assigned automatically to connect the terminals and pins.

For more information about the how to set the "INSTRUMENT\_CARD" mode and "ROW\_COLUMN" mode, refer to Hardware Configuration.

Example:

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

## conpth

---

Purpose: Clear all connections and adds new terminal-pin connections by path.

Format:

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

`instMTRX` The matrix name in hardware configuration, it's optional.

`ter` The list of terminals to be connected.

`pin` The list of pins to be connect.

`row` The row used to connect terminals and pins

Remarks: All terminals and pins will be connected together by the assigned row.

One command cannot connect two paths.

One command cannot connect paths in multiple matrixs.

You can also connect only terminals or only pins by this function. But when connecting only terminals, `instMTRX` is required, otherwise the function does not know which instrument to send the command to.

Example: 70X

```
conpth(MTRX1, SMU1, 1, 'A')
conpth(SMU1, 1, 'A')
conpth(SMU1H, 1, 'A')
conpth(SMU1L, 1, 'A')
conpth([SMU1, SMU2], [1, 2], 'A')
conpth(MTRX1, [SMU1, SMU2], [], 'A')
conpth([], [1, 2], 'A')
```

Series 3700 System Switch

```
conpth(MTRX1, SMU1, 1, '1')
conpth(SMU1, 1, '1')
conpth(SMU1H, 1, '1')
conpth(SMU1L, 1, '1')
conpth([SMU1, SMU2], [1, 2], '1')
conpth(MTRX1, [SMU1, SMU2], [], '1')
conpth([], [1, 2], '1')
```

## delay

---

Purpose: Provides user-programmable delay within a test sequence. The units are in milliseconds.

Format:

```
delay(iDelayTime)
```

## delcon

---

**Purpose:** Delete terminal-pin connections.

**Format:**

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

**instMTRX** The matrix name in hardware configuration, it's optional.

**ter** The list of terminals to be connected.

**pin** The list of pins to be connect.

**more\_pin** Indicates more pins to connect.

**Remarks:** Normally delcon() supports "ROW\_COLUMN" mode for a matrix. For more information about the how to set the "INSTRUMENT\_CARD" mode and "ROW\_COLUMN" mode, refer to ACS Example hardware configurations.

**Example:**

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



## delpth

---

**Purpose:** Delete terminal-pin connections by specified path.

**Format:**

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

**instMTRX** The matrix name in hardware configuration, it's optional.

**ter** The list of terminals to be disconnected.

**pin** The list of pins to be disconnected.

**row** The row used to connect the terminals and pins.

**Remarks:** Note that the ter-pin-row has to be the actual group when they are connected, otherwise there is no action in the Matrix.

**Example: 70X**

```
delpth(MTRX1, SMU1, 1, 'A')
delpth(SMU1, 1, 'A')
delpth(SMU1H, 1, 'A')
delpth(SMU1L, 1, 'A')
delpth([SMU1, SMU2], [1, 2], 'A')
delpth(MTRX1, [SMU1, SMU2], [], 'A')
delpth([], [1, 2], 'A')
```

**Series 3700 System Switch**

```
delpth(MTRX1, SMU1, 1, '1')
delpth(SMU1, 1, '1')
delpth(SMU1H, 1, '1')
delpth(SMU1L, 1, '1')
delpth([SMU1, SMU2], [1, 2], '1')
delpth(MTRX1, [SMU1, SMU2], [], '1')
delpth([], [1, 2], '1')
```

## devclr

---

**Purpose:** Sets all sources to a zero state.

**Format:**

```
devclr(*args)
```

**Example:**

```
devclr()
devclr(SMU1)
devclr(SMU1, CVU1)
```

**Remarks:** This function will send output off commands or call the Model 4200 devclr function. It will not work on a matrix. If the system is configured using KCON the Model 4200 devclr function will execute. This function will clear all sources sequentially. Prior to clearing all Keithley Instruments supported instruments, GPIB based instruments will be cleared by sending all strings defined with kibdefclr. Devclr is implicitly called by clrcon, devint, execut, and tstdsl.

## devint

**Purpose:** Resets the instruments and clears the system by opening all relays and disconnecting the pathways. Meters and sources are reset to the default states. Refer to the specific hardware manuals for listings of the default conditions and ranges for the instrumentation.

**Format:**

```
devint(*args)
```

**Example:**

```
devint()
devint(SMU1)
```

**Remarks:** This function will send reset commands or call the Model 4200 devint function. If the system is configured using KCON the Model 4200 devclr function will execute. The Model 4200 devclr function will execute as follows:

This function will reset all instruments in the system to their default states.

This function will preform the following actions prior to resetting the instruments:

1. Clear all sources by calling devclr.
2. Clear the matrix cross-points by calling clrcon.
3. Clear the trigger tables by calling clrtrg.
4. Clear the sweep tables by calling clrscn.
5. Reset GPIB instruments by sending the string defined with kibdefint.
6. Stop the pulse generator card, and click the standard pulse mode and its default settings (like \*RST). devint is implicitly called by execut and tstdsl.

## NOTE

The following table indicates the default settings for the Series 2600B and 2400 instruments after using the devint LPT command.

Default settings		
Series 2600B SourceMeter instruments setting after devint		
Output		turn source off
Reset		reset all bits of following register to 0: <ul style="list-style-type: none"> <li>• standard event register</li> <li>• operation event register</li> <li>• measurement event register</li> <li>• questionable event register</li> </ul>
Long-form and short-form versions		command word will be sent in short-form version
ACS hardware configuration setting window	if interlock enabled	enable interlock
	if rear panel enabled	enable rear panel
	if beeper disabled	disable beeper

<b>Default settings</b>	
<b>Series 2400 SourceMeter instruments setting after devint</b>	
Current range	0.1 A (all SMUs)
Output	clear
Error queue	clear
Status model	reset all bit
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
Reset	each instruments default factory setting

## disable

---

Purpose: Stops the timer and sets the time value to zero. Timer reading is also stopped.

Format:

```
disable(unitname)
```

unitname:           The instrument name of timer module ('in  
\\ACSKATS\CONFIG\ACS\_hdcon\_Online.kcf').

Example:

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

## enable

---

Purpose: Provides correlation of real time to measurements of voltage, current, conductance, and capacitance.

Format:

```
enable(unitname)
```

unitname:           The instrument ID of timer module (TIMERn).

Example:

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

---

**execut**

---

Purpose: Causes system to wait for the preceding test sequence to be executed.

Format:

```
execut(*args)
```

Example:

```
execut()  
execut(SMU1)
```

Remarks: For 42xx or Series 2600B SourceMeter instruments, this function will wait for all previous LPT commands to complete and then will issue a devint.

---

**forcei/forcev**

---

Purpose: Programs a sourcing instrument to generate a voltage or current at a specific level.

Format:

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

unitname:           The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

dValue:             The level of the bipolar voltage or current forced in volts or amperes

---

**get common**

---

Purpose: Get common attribute from global\_dict. Return key list: [UNITLIST, PLC, pin]

Format:

```
getcommon()
```

Example:

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

## getinstattr

Purpose: Get the instrument attribute from the instrument name string.

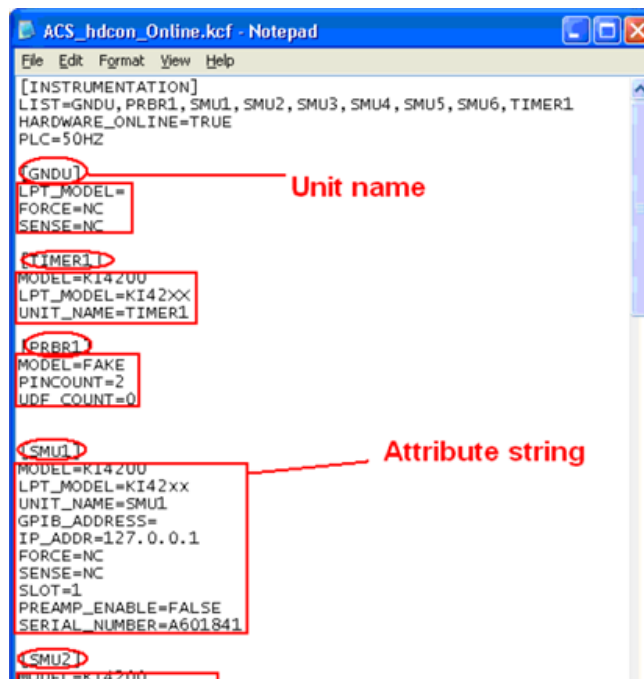
Format:

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

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

attr\_str: The attribute string list in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf for a instrument name.

**Figure 22: 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")
KI4200
```

---

**getinstid**

---

Purpose: Get the instrument identifier (ID) from the instrument name string, only used in 42xx.

Format:

```
getinstid(unitname)
```

unitname:           The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

return value:

```
instrument identifier (ID)
```

Example:

```
print getinstid(SMU1)
4100
```

---

**getstatus**

---

Purpose: Returns the operating state of the desired instrument. This function only used on 42xx.

Format:

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

unitname\_str:    The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

iCode:            The parameter of query.

return value:    The data returned from the instrument. `getstatus` returns one item.

Remarks:

Valid Errors:

The UT\_INVLDPRM invalid parameter error is returned from `getstatus`. The status item parameter is illegal for this device. The requested status code is invalid for selected device.

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

<b>Getstatus parameters</b>		
<b>iCode</b>	<b>Comment</b>	
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 limitX). 4 = RANGE (at the top of the range set by rangeX)
KI_RANGE_COMPLIANC E	Active compliance status for fixed range	In range compliance if 1
KI_COMPLNC_EVER	Compliance history	Reset by reading compliance history and by devint

<b>ki20xxlpt Getstatus parameters</b>	
<b>iCode</b>	<b>Comment</b>
KI_VPRANGE	The presently programmed voltage range
KI_VARANGE	The presently active voltage range
KI_VMRANGE	The range used when the last measurement was performed. For autorange, the range at which the previous V measurement was performed

Valid Errors:

The UT\_INVLDPRM invalid parameter error is returned from getstatus. The status item parameter is illegal for this device. The requested status code is invalid for selected device.

Example:

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

## **gpibenter**

---

**Purpose:** Used to read a device dependent string from an instrument connected to the GPIB interface.

**Format:**

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

**unitname:** (string type)The instrument name in C:\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**max\_size:** A value specifying the maximum number of characters you want to receive. maxlength can be a number from 0 to 32767 (hex FFFF).

**Return value:** (tuple type)(receive str, length, status)or error code

**Example:**

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

## **gpibsend**

---

**Purpose:** Sends a device dependent command to an instrument connected to the GPIB interface.

**Format:**

```
gpibsend(unitname, cmd_str)
```

**unitname:** The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**cmd\_str:** A string to be sent to the device. Note: Terminating character(s) are automatically added to the end of this string when it is sent. The default terminator is a line feed character.

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

**Example:**

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

## **gpibspl**

---

**Purpose:** A serial poll reads the status of an instrument connected to the GPIB interface.

**Format:**

```
gpibspl(unitname)
```

**unitname:** The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**Return value:** Tuple - (receive number, status) or error code

**Example:**

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



---

## imeast

---

Purpose: Force a read of the timer and return the result.

Format:

```
imeast(unitname)
```

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf. For example, TIMER1, TIMER2

Return value: Elapsed time from enable(TIMER1).

Remarks: This command applies to all timers. Must call enable(TIMERn) first.

Example:

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

---

## intgi/intgv

---

Purpose: Performs voltage or current measurements averaged over a user-defined period (usually, one AC line cycle). This averaging is done in hardware by integration of the analog measurement signal over a period of specified time. The integration is automatically corrected for 50 or 60Hz power mains.

Format:

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

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

Return value: Result data

Example:

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

---

## limiti/limitv

---

Purpose: Allows the programmer to specify a current or voltage limit other than the instrument's default limit.

Format:

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

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

dValue: The maximum level of the current or voltage. The value is bidirectional. For example, a limitv ("SMU1", 10.0) limits the voltage of the current source of SMU1 to 10.0 V. A limiti ("SMU1", 1.5E-3) limits the current of the voltage source of SMU1 to 1.5 mA.

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

## lorangei/lorangev

---

**Purpose:** Defines the bottom autorange limit.

**Format:**

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

**unitname:** The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**dValue:** The value of the desired instrument range, in volts or amperes.

**Remarks:** `lorange` is used with autoranging to limit the number of range changes and therefore saves test time.

For the 42xx, if the instrument was on a range lower than the one specified by `lorange`, the range is changed. Model 4200-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 `lorangeX` executes.

For the 23x instruments, works as auto range. The second `dValue` will be ignored.

It cannot be used for the Series 2400 SourceMeter instruments.

**Example:**

```
lorangei(SMU1, 2.0E-6)
```

## measi/measv

---

**Purpose:** Allows the measurement of voltage, current.

**Format:**

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

**unitname:** The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

**Return value:** Result data

**Remarks:** For this measurement, the signal is sampled for a specific period of time. This sampling time for the measurement is called the integration time. For the `measX` function, the integration time is fixed at 0.01PLC. For 60Hz line power, 0.01PLC = 166.67 $\mu$ s (0.01PLC/60Hz). For 50Hz line power, 0.01PLC = 200 $\mu$ s (0.01PLC/50Hz).

**Example:**

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

## measz

**Purpose:** Performs an impedance measurement on a CVU or other capacitance measuring instrument.

**Format:**

```
measz(unitname, iModel, iSpeed)
```

**unitname:** (string type) The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf. Only “CVUn” and “CMRTn” are supported.

**iModel:** Measurement Model (see next Figure Table 10–12).

**iSpeed:** Measure speed: KI\_CVU\_SPEED\_FAST, KI\_CVU\_SPEED\_NORMAL, or KI\_CVU\_SPEED\_QUIET

**Return value:** [result1, result2]

**result1:** The first result data of the selected measure Model

**result2:** The second result data of the selected measure Model

**Remarks:** The measurement models are listed in the next Table.

**Measurement speed settings:** KI\_CVU\_SPEED\_FAST performs fast measurements (higher noise)

Measurement settings			
unitname_str (Model name)	iModel (Measurement Model)		parameter values
CVU1	ZTH	Impedance (Z) and phase ( $\theta$ 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 ( $\theta$ 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
	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	

Measurement settings			
unitname_str (Model name)	iModel (Measurement Model)		parameter values
	LS-Rs	Series inductance value and equivalent resistance	KI_AGCV_TYPE_LSRS
	Z-thd	Impedance (Z) and phase ( $\phi$ in degree)	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 degree)	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)
```

## rangei/rangev

Purpose: Selects a measurement range and prevents the selected instrument from autoranging. By selecting a range, the time required for autoranging is eliminated.

Format:

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

unitname\_str: (string type) The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

dvalue: The value of the highest measurement to be taken. The most appropriate range for this measurement will be selected. If range is set to 0, the instrument will autorange except Series 2600B SourceMeter instruments.

Example:

```
rangei(SMU1, 2.0E-3) # Click current range of 2mA.
```

## rdelay

Purpose: A user-programmable delay in seconds.

Format:

```
rdelay(dDelayTime)
```

n: The desired delay duration in seconds.

Example:

```
rdelay(0.02)# Pause for 20ms
```

---

**setauto**

---

Purpose: Re-enables autoranging and cancels any previous rangeX command for the specified instrument.

Format:

```
setauto(unitname)
```

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf.

Remarks: When an instrument is returned to the autorange mode, it will remain in its present range for measurement purposes. The source range will change immediately.

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

Example:

```
setauto(SMU1) # Enable autorange mode.
```

---

**setfreq**

---

Purpose: A CV test command. Sets the frequency for the AC drive.

Format:

```
setfreq(unitname,dFreq)
```

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf. Only "CVUn" and "CMRTn" support.

frequency: Frequency of the AC drive in Hz

Example:

```
status = setfreq(CVU1,10000)
```

---

**setlevel**

---

Purpose: A CV test command. Sets the AC drive voltage level.

Format:

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

unitname: The instrument name in \\ACS\KATS\CONFIG\ACS\_hdcon\_Online.kcf. Only "CVUn" and "CMRTn" are supported.

dSignalLevel: Voltage level of the AC drive (10mV to 100mVRMS) in volts. Different valid ranges for CVU and CMTR

Example:

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

## **setmode**

---

Purpose: Set instrument specific operating mode parameters.

Format:

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

Remarks: Setmode allows control over certain instrument specific operating characteristics. Refer to the specific instrument's documentation for more information on what each instrument supports. Refer to the next tables for more information regarding modifier values.

Setmode			
ki23x1pt Parameters			Comment
unitname_str	iModifier	dValue	
SMU1	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_SENSE	KI_SENSE_LOCA (or 0) KI_SENSE_REMO( or 1)	Set remote, local, sense mode: KI_SENSE_LOCA: Clicks Local Sense (2-wire). KI_SENSE_REMO: Clicks Remote Sense (4-wire).
	KI_TRIG_IN	KI_TRIG_IN_CONT = 0 KI_TRIG_IN_SRC = 1 KI_TRIG_IN_DLY = 2 KI_TRIG_IN_SRCDLY = 3 KI_TRIG_IN_MSR = 4 KI_TRIG_IN_SRCMSR = 5 KI_TRIG_IN_DLYMSR = 6 KI_TRIG_IN_SRCDLYMSR = 7 KI_TRIG_IN_PULSE = 8	Input triggers. Input trigger are used to control when source, delay, and measure operations occur: KI_TRIG_IN_CONT: Continuously process all SDM(Source Delay Measure) cycles. KI_TRIG_IN_SRC: Each trigger will process an SDM cycle. KI_TRIG_IN_DLY: Initial trigger sets source. Each subsequent trigger initiates a delay and measure then sets source of next SDM cycle. KI_TRIG_IN_SRCDLY: Two trigger process each SDM cycle. First trigger sets source. Second trigger initiates a delays and measure. KI_TRIG_IN_MSR: Initial trigger sets source and causes a delay. Second trigger initiates measure, and then, for next SDM cycle, sets source and initiates a delay. KI_TRIG_IN_SRCMSR: Two triggers process each SDM cycle. First trigger sets source and initiates a delay. Second trigger initiate a measure. KI_TRIG_IN_DLYMSR: Initial trigger sets source. Two triggers process each SDM cycle. First trigger initiates a delay. Second trigger initiates a measure and sets source of nest SDM cycle. KI_TRIG_IN_SRCDLYMSR: Three triggers process each SDM cycle. First trigger sets source. Second trigger initiates a delay. Third trigger initiates a measure. KI_TRIG_IN_PULSE: Pulse sweep trigger. Each trigger process the on the time and off time of each pulse in the sweep. Two measurements are made on each pulse.

<b>Setmode</b>			
<b>ki23x1pt Parameters</b>			<b>Comment</b>
	KI_TRIG_SOURCE	KI_TRIG_X = 0 KI_TRIG_GET = 1 KI_TRIG_TALK = 2 KI_TRIG_EXTERNAL = 3 KI_TRIG_INTERNAL = 4	Input trigger origin. The input trigger stimulus may be provided by front manual trigger function, and external device that applies a TTL level pulse to the TRIGGER in connector on the rear panel, or an appropriate IEEE-488 operation. KI_TRIG_X: IEEE X origin. "X" sent over IEEE-488 bus. KI_TRIG_GET: Group execute trigger KI_TRIG_TALK: Unit address to talk over IEEE-488 bus. KI_TRIG_EXTERNAL: Negative going TTL level pulse applied to TIRIGGER in connector. KI_TRIG_INTERNAL: Front panel MANUAL trigger function or HO command over IEEE-488 bus.
	KI_TRIG_OUT	KI_TRIG_OUT_NONE = 0 KI_TRIG_OUT_SRC = 1 KI_TRIG_OUT_DLY = 2 KI_TRIG_OUT_SRCDLY = 3 KI_TRIG_OUT_MSR = 4 KI_TRIG_OUT_SRCMSR = 5 KI_TRIG_OUT_DLYMSR = 6 KI_TRIG_OUT_SRCDLYMSR = 7 KI_TRIG_OUT_PULSE = 8	Output trigger generation: KI_TRIG_OUT_NONE: No output triggers. KI_TRIG_OUT_SRC: Output trigger pulse after every source phase. KI_TRIG_OUT_DLY: Out put trigger pulse after every delay phase. KI_TRIG_OUT_SRCDLY: Out put trigger pulse after every source phase and delay phase. KI_TRIG_OUT_MSR: Out put trigger pulse after every source phase and measure phase. KI_TRIG_OUT_SRCMSR: Output trigger pulse after every source phase and measure phase. KI_TRIG_OUT_DLYMSR: Out put trigger pulse after every delay phase and measure phase. KI_TRIG_OUT_SRCDLYMSR: Out put trigger pulse after every source phase, delay phase and measure phase. KI_TRIG_OUT_PULSE: For pulse sweeps . Output trgger pulse after end of each off time measure.
	KI_SWEEPEND_TRIGOUT	KI_SWEEPEND_TRIGOUT_EN = 1 KI_SWEEPEND_TRIGOUT_DIS = 0	When enable, an output trigger pulse occurs at the end of the sweep.
	KI_AVGNUMBER	0, 2, 4, 8, 16, 32	Number of readings to take average. 0 means disable average filter



Setmode			
ki20xxlpt parameters			Comment
Unitname_str	iModifier	dValue	
VMTR1	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time will be used for the intgv command.
	KI_AVGMODE	KI_MEASX KI_INTEGRATE	Controls what kind of readings are taken for avgv calls.

Setmode			
kiSeries 2400 instruments lpt parameters			Comment
Unitname_str	iModifier	dValue	
SMU1	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 0.01~10(DC) and 0.004~0.1(2430 Pulse mode)
SMU1 (only 2430 SMU)	KI_TRIG_IN_CONT	<value>	Set the output pulse count.
SMU1	PULSE_MODE_PULSE	VOLT CURR	Click pulse mode and pulse source function VOLT: voltage source VOLT: voltage source CURR: current source
	PULSE_MODE_WIDTH	<value>	Click pulse mode and set pulse width
	PULSE_MODE_DELAY	<value>	Click pulse mode and set pulse delay

Setmode			
kiSeries 2600B instruments lpt parameters			Comment
unitname_str	iModifier	dValue	
SMU1	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 KI_INTEGRATE	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 call.
	KI_OFFMODE	KI_OFF_NORM KI_OFF_ZERO KI_OFF_OPEN	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_SENSE	KI_SENSE_LOCAL KI_SENSE_REMOTE KI_SENSE_CALIBRATION	Set remote, local, sense mode, or calibration. KI_SENSE_LOCAL: Clicks Local Sense (2-wire). KI_SENSE_REMOTE: Clicks Remote Sense (4-wire). KI_SENSE_CALIBRATION: Clicks calibration sense mode.

<b>Setmode</b>			
<b>ki42cvulpt parameters</b>			<b>Comment</b>
<b>unitname_str</b>	<b>iModifier</b>	<b>dValue</b>	
	KI_CVU_COMPENSATE	[1,1,1] To each item: 0=OFF 1=ON	Set open_comp, short_comp, load_comp in one command. Value must be list type and have three items.
CVU1	KI_CVU_CABLE_CORRECT	0,1.5 or 3	Cable length setting (in Meters), Can be set to any floating point number between 0 and 3.0, but will be coerced to 0, 1.5 or 3.
	KI_CVU_CUST_SPEED	[delay_factor, filter_factor, aperture] = [1,1,1] delay_factor: 0~100, aperture: 0006~10.002	Select customize speed mode and set parameter value, including delay_factor, filter_factor, and aperture. Value must be list type and have three items.
	KI_CVU_OPEN_COMPENSATE KI_CVU_SHORT_COMPENSATE KI_CVU_LOAD_COMPENSATE	0=OFF 1=ON	Enable or disable compensation constants for open load and short.
	KI_CVU_FILTER_FACTOR	0 to 100	Set 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	Set 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 type (see measz).
	KI_CVU_CHANNEL	1~8	Selects CVU card on which subsequent card CVU commands will act.
	KI_CVU_OFFSET	-30~30	Apply 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 HCUR/HPOT.
	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.
KI_CVU_MODE	0 or 1	0: set CVU to user mode 1: set CVU to system mode	

<b>Setmode</b>			
<b>ki4284lpt parameters</b>			<b>Comment</b>
<b>unitname _str</b>	<b>iModifier</b>	<b>dValue</b>	
CMTR1	KI_CVU_CABLE_CORRECT	0,1.5 or 3	Cable length setting (in Meters), Can be set to any floating point number between 0 and 3.0, but will be coerced to 0, 1.5 or 3.
	KI_CVU_OPEN_COMPENSATE KI_CVU_SHORT_COMPENSATE KI_CVU_LOAD_COMPENSATE	0=OFF 1=ON	Enable or disable compensation constants for open load and short.
	KI_CVU_FILTER_FACTOR	0 to 100	Set 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	Set 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 type see measz.
	KI_CVU_MODE	0 or 1	0: set CVU to user mode 1: set CVU to system mode
	KI_AGCV_CORRECT_METHOD	KI_AGCV_CORRECT_MET HOD_MULT = 0 KI_AGCV_CORRECT_MET HOD_SING = 1	elects the correction mode (Single or Multi). Scanner I/F should be installed for the Multi mode KI_AGCV_CORRECT_METHOD_SING : Sets the correction mode to "SINGLE" KI_AGCV_CORRECT_METHOD_MULT : Sets the correction mode to "MULTI"
	KI_AGCV_TRIG_SOURCE	KI_AGCV_TRIG_SOURCE KI_AGCV_TRIG_INTERNAL = 0 KI_AGCV_TRIG_HOLD = 1 KI_AGCV_TRIG_EXTERNAL = 2 KI_AGCV_TRIG_BUS = 3	Selects the trigger mode: KI_AGCV_TRIG_INTERNAL: Sets trigger source to "internal" KI_AGCV_TRIG_HOLD: Sets trigger source to "manual" KI_AGCV_TRIG_EXTERNAL: Sets trigger source to "external connector on the rear panel" KI_AGCV_TRIG_BUS: Sets trigger source to "GPIB/LAN/USB"
	KI_AGCV_INIT_CONTINUE	0=OFF 1=ON	Enables the automatic trigger to change state from the "Idle" state to the "Wait for Trigger" state. Refer to the chapter on Remote Control ON or 1: Enables automatic trigger state change OFF or 0 (Preset value): Disables automatic trigger state change

Setmode			
ki4284lpt parameters			Comment
	KI_AGCV_DISPLAY_PAGE  KI_AGCV_DISPLAY_MEAS = 0 KI_AGCV_DISPLAY_BNUMBER = 1 KI_AGCV_DISPLAY_BIN_NUMBER = 2 KI_AGCV_DISPLAY_BIN_COUNT = 3 KI_AGCV_DISPLAY_SWEEP = 4 KI_AGCV_DISPLAY_CORRECTION = 5 KI_AGCV_DISPLAY_MEAS_SETUP = 6 KI_AGCV_DISPLAY_CORRECTION_SETUP = 7 KI_AGCV_DISPLAY_LIMIT_TABLE_SETUP = 8 KI_AGCV_DISPLAY_SWEEP_SETUP = 9 KI_AGCV_DISPLAY_SELF_TEST = 10 KI_AGCV_DISPLAY_MLARGE = 11 KI_AGCV_DISPLAY_SYSTEM_INFORMATION = 12 KI_AGCV_DISPLAY_SERVICE = 13		Selects the page to be displayed. KI_AGCV_DISPLAY_MEAS: Sets displayed page to <MEAS DISPLAY> KI_AGCV_DISPLAY_BNUMBER: Sets displayed page to <BIN No. DISPLAY> KI_AGCV_DISPLAY_BIN_COUNT: Sets displayed page to <BIN COUNT DISPLAY> KI_AGCV_DISPLAY_SWEEP: Sets displayed page to <LIST SWEEP DISPLAY> KI_AGCV_DISPLAY_MEAS_SETUP: Sets displayed page to <MEAS SETUP> KI_AGCV_DISPLAY_CORRECTION_SETUP: Sets displayed page to <CORRECTION> KI_AGCV_DISPLAY_LIMIT_TABLE_SETUP: Sets displayed page to <LIMIT TABLE SETUP> KI_AGCV_DISPLAY_SWEEP_SETUP: Sets displayed page to <LIST SWEEP SETUP> KI_AGCV_DISPLAY_CATALOG: Sets displayed page to <CATALOG> KI_AGCV_DISPLAY_SYSTEM_INFORMATION: Sets displayed page to <SYSTEM INFORMATION> KI_AGCV_DISPLAY_SELF_TEST: Sets display page to <SELF TEST> KI_AGCV_DISPLAY_MLARGE: Sets page to display measurement results in large characters KI_AGCV_DISPLAY_SCONFIG: Sets displayed page to <SYSTEM CONFIG> KI_AGCV_DISPLAY_SERVICE: Sets displayed page to <SERVICE>

Setmode				
Support	Parameters			Comment
	instr_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>	Number of readings to take when KI_TRIGMODE is set to KI_AVERAGE.
		KI_AVGTIME	<value> (in units of seconds)	Time between readings when KI_TRIGMODE is set to KI_AVERAGE.

Setmode				
Support	Parameters			Comment
No-Op (accepted but not responded to)		KI_MX_DEFMODE	KI_HIGH KI_LOW	Sets the default matrix mode to high current mode or low current mode. This setting will remain in effect until the end of the current session and is not reset by devint.
		KI_HICURRENT	KI_ON	Forces the matrix into high current mode. The mode will revert to the default at the next devint unless the configuration file sets this parameter to reset on a clrcon.
		KI_CC_AUTO	KI_ON KI_OFF	Turns automatic compliance clear processing on or off. devint will reset 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 after a source value change and it takes the circuit under test to settle and prevent false compliance detection due to transients.
		KI_CC_COMP_DLY	<value>	The time between compliance scans while processing compclr. This also represents the time after a source value change takes the circuit under test to settle and prevent false compliance detection due to transients, but 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 level desired for the subsequent measurements.
Supported	SMUn	KI_INTGPLC	<value> (in units of line cycles)	Specifies the integration time the SMU will use for the intgx and singtx commands. The default devint value is 1.0. The valid range is 0.01 to 10.0.
		KI_AVGMODE	KI_MEASX KI_INTEGRATE	Controls what kind of readings are taken for avgX calls. The devint default value is KI_MEASX. When KI_INTEGRATE is specified, the integration time used is that specified by the KI_INTGPLC setmode call.
No-Op (accepted but not responded to)		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 will turn 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 = 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.

Setmode			
Support	Parameters		Comment
	KI_LIM_INDCTR	Any	Controls what measure value 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 signal being measured.
	KI_LIM_MODE	KI_INDICATOR KI_VALUE	Controls whether SMU will return an indicator value when in limit or overrange, or the actual value measured. The default mode after a devint is to return an indicator value.
	KI_RANGE_DELAY	<value> (in seconds) ranges from -2147493.647 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 preamp range. Value may be negative to shorten rather than lengthen the overall range change delay. In no event will the overall delay time be less than the preamp circuit hardware switching time. The devint default value is 0.0.
	KI_RANGE_SETTLE	0.01 0.1 1.0 2.5 5.0 10.0	Controls how long the SMU driver will delay when changing a preamp range. Value is specified in percent settling accuracy, although at present, only six specific values are valid. The actual delay time depends on which range the preamp is being switched from and which range it is being switched to. The devint default value is 1.00
	KI_VMTR		Sets the SMU as a volt meter. The ranges used are representative of the type of instrument being simulated. NOTE: this setmode will turn 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. 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.

## NOTE

These modifiers perform no operations in the Model 4200-SCS. They are included only for compatibility, so that existing S600 programs using the setmode function can be ported to the Model 4200-SCS without upsets.

Example:

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

---

**smeasz\_sweepv**

---

**Purpose:** Performs and returns CD measurements for a voltage sweep with specified frequency bias. Posts data after the sweep is completed.

**Format:**

```
smeasz_sweepv(unitname, iSpeed, dVStart, dVStop, iStepNum, dDelayTime)
```

**return value:** [rvalue1, rvalue2]

**result1:** List of the first result of the selected measure Model

**result2:** List of the second result of the selected measure Model

**Example:**

```
smeasz_sweepv(CVU1, KI_CVU_SPEED_FAST, -3, 3, 10, 0.01)
```

---

**srangei/srangev**

---

**Purpose:** Clicks the current/voltage source range and prevents the selected instrument from auto-ranging. By clicking a range, the time required for auto-ranging is eliminated.

**Format:**

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

---

**tstsel**

---

**Purpose:** Used to enable or disable a test station. Only used for the Model 4200.

To relinquish control of an individual test station, a new test station must now be selected using `tstsel` before any subsequent test control functions are run. The `tstdsl` command has the same effect as the `tstsel (0)` command.

**Format:**

```
tstsel(iStatus = 1)
```

**Remarks:** `tstsel` is normally called at the beginning of a test program.

## PTM examples

### ACSLPT using example: vgsid1

```

##outputlist=GateV,DrainI,Time##
from ACS_PostData import *
from ACSLPT import *
from ptmplt.constantlpt import *
from math import *
Get4200HWCtrl()
def vgsid1(DrainSMU, DrainPin, GateSMU, GatePin, SourceSMU, SourcePin, BulkSMU,
BulkPin, GateVStart, GateVStop, numberofpoint, SweepDelay, DrainV,
SourceV, BulkV, RangeDrainI, ComplianceDrainI, StoponCompliance, NPLC):
    GateV=[]
    DrainI=[]
    Time_meas=[]
    tstsel(1)
    #Some input checking is needed
    if GateVStart < -200 or GateVStart > 200:
        return INVALID_PARAM
    if GateVStop < -200 or GateVStop > 200:
        return INVALID_PARAM
    if numberofpoint < 1 or numberofpoint > 4096:
        return INVALID_PARAM
    if SweepDelay < 0 or SweepDelay > 100:
        return INVALID_PARAM
    if DrainV < -200 or DrainV > 200:
        return INVALID_PARAM
    if SourceV < -200 or SourceV > 200:
        return INVALID_PARAM
    if BulkV < -200 or BulkV > 200:
        return INVALID_PARAM
    if RangeDrainI < 1 or RangeDrainI > 12:
        return INVALID_PARAM
    if ComplianceDrainI < -0.1 or ComplianceDrainI > 0.1:
        return INVALID_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)
if RangeDrainI == 1:
    setauto(DrainSMU)
elif RangeDrainI == 2:
    lorangei(DrainSMU, 1e-11)
elif RangeDrainI == 3:
    lorangei(DrainSMU, 1e-10)
elif RangeDrainI == 4:
    lorangei(DrainSMU, 1e-9)
elif RangeDrainI == 5:
    lorangei(DrainSMU, 1e-8)
elif RangeDrainI == 6:
    lorangei(DrainSMU, 1e-7)
elif RangeDrainI == 7:
    lorangei(DrainSMU, 1e-6)
elif RangeDrainI == 8:
    lorangei(DrainSMU, 1e-5)
elif RangeDrainI == 9:
    lorangei(DrainSMU, 1e-4)
elif RangeDrainI == 10:
    lorangei(DrainSMU, 1e-3)
elif RangeDrainI == 11:
    lorangei(DrainSMU, 1e-2)
elif RangeDrainI == 12:
    lorangei(DrainSMU, 1e-1)
```

```
# auto range
# limited auto 10pA
#limited auto 100pA
#limited auto 1nA
#limited auto 10nA
#limited auto 100nA
#limited auto 1uA
# limited auto 10uA
# limited auto 100uA
# limited auto 1mA
# limited auto 10mA
# limited auto 100mA
```

```

        lorangei(DrainSMU, 0.1)
    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)
            print GateV_tmp
            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
#####CALL#####
DrainSMU=SMU1
DrainPin=1
GateSMU=SMU2
GatePin=2
SourceSMU=GNDU
SourcePin=3
BulkSMU=GNDU
BulkPin=4
GateVStart=0.0

```

```
GateVStop=3.0
numberofpoint=21
SweepDelay=0.001
DrainV=0.1
SourceV=0
BulkV=0
RangeDrainI=1
ComplianceDrainI=0.1
StoponCompliance=0
NPLC=1
vgsidl(DrainSMU, DrainPin, GateSMU, GatePin, SourceSMU, SourcePin, BulkSMU,
      BulkPin, GateVStart, GateVStop, numberofpoint, SweepDelay, DrainV, SourceV,
      BulkV, RangeDrainI, ComplianceDrainI, StoponCompliance, NPLC)
```

---

## Python Test Module (PTM) Debug Tool

### In this section:

---

PTM debug tool introduction .....	3-1
PythonWin description .....	3-2
PTM debugging.....	3-4
Debug tool limitations.....	3-20

### PTM debug tool introduction

Automated Characterization Suite (ACS) software provides you with a debug tool that can be used for test modules that use scripts, such as the python test modules (PTM). The name of the tool is PythonWin Debugger and it is integrated in ACS software for your convenience.

PythonWin Debugger will help you to debug your PTM, step-by-step, and can monitor your script variables during test execution. The PythonWin Debugger helps you refine and optimize your PTM scripts and assists in tracking the testing process.

#### NOTE

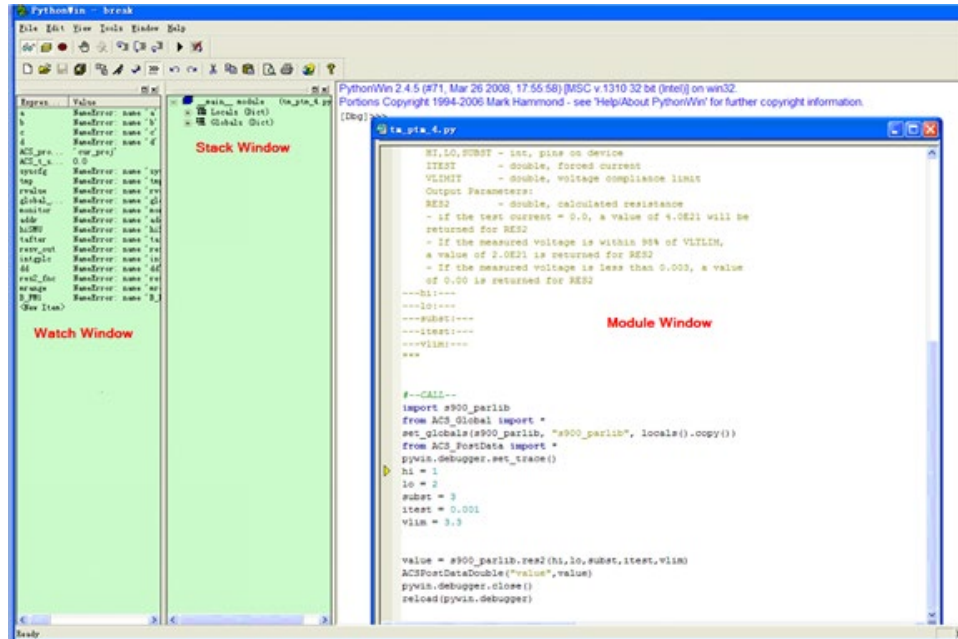
Python is Copyright (c) 2001-2010 Python Software Foundation. All Rights Reserved.

Pythonwin - Python IDE and GUI Framework for Windows. Copyright 1994-2010 Mark Hammond (mhammond@skippinet.com.au). All Rights Reserved.

## PythonWin description

PythonWin Debugger is a graphical user interface (GUI) and includes an easy to use interactive editing environment (see next Figure).

Figure 23: Pythonwin GUI



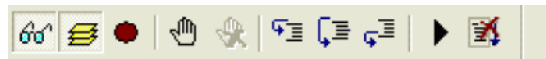
PythonWin Debugger supports setting conditional breakpoints, single stepping at the source line level, inspects stack frames, source code listings, and evaluates arbitrary Python code in the context of any stack frame. It also supports postmortem debugging for you and can be called under program control.

The following is an overview on how PythonWin functions (see next Figure):

### NOTE

The list of functions below (1 - 10) are represented in the next Figure by viewing the ten icons from left to right.

Figure 24: Pythonwin debug toolbar



**Debugging Toolbar icon descriptions:**

1. Watch (monitor the modules that you insert in the Watch window)
2. Stack view (view the modules and global variables in the Stack view window)
3. Breakpoint list (inserts designated breakpoints and a dialog box opens to view the condition and location of each)
4. Toggle Breakpoint (add or remove a breakpoint)
5. Clear All Breakpoints (removes the breakpoints)
6. Step (steps into the current module source code statement and executes a single-step line-by-line)
7. Step over (steps over the current module code, execute, and continue)
8. Step out (steps out of the current module code)
9. Go (continue execution)
10. Close (cancels the debugging session)

**NOTE**

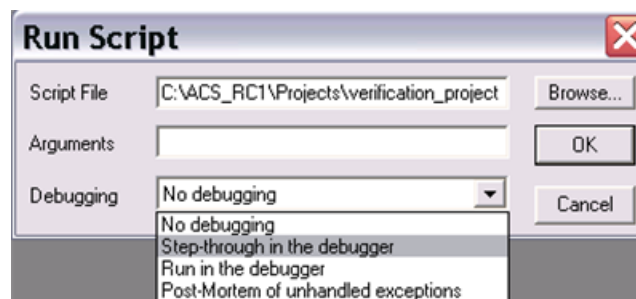
The list of functions below (1 - 4) are represented in the next Figure by the red box that is around the four icons from left to right.

**Figure 25: Pythonwin Standard Toolbar****Standard Toolbar icon descriptions:**

1. Import/Reload (shows the condition and location of a Python module)
2. Run (run a Python script by choosing the Script File, Arguments, and the Debugging required)(see next Figure)

**NOTE**

You must click the down-arrow to choose the type of debugging desired when running a script.

**Figure 26: PythonWin Run Script**

3. Check (checks the current file without executing it)
4. Interactive window (show or hide the interactive window)

## PTM debugging

Figure 27: VARIABLE - NOTE

**NOTE**

- You cannot run the debug tool if ACS is in Demo mode.
- You cannot run the debug tool for Automation testing.
- You cannot run the debug tool on instruments that are connected through LXI.
- You cannot run the debug tool on python test modules if a LPT module is included.

### Enable debug tool

**NOTE**

If you do not see the debug icon in the toolbar, you need to make sure that debug has been enabled. The Debug Run icon is circled in the figure below (see next Figure).

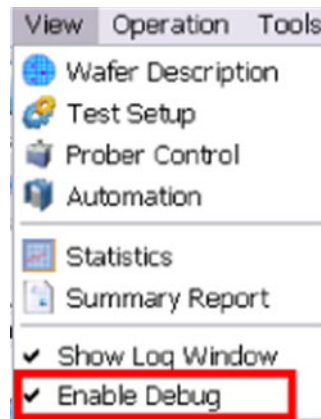
Figure 28: Debug icon in toolbar



**NOTE**

The default state of the debug tool is disabled. To enable the debug option, you must select the View option in the main toolbar, and in the drop-down list click the Enable Debug function (see next Figure).

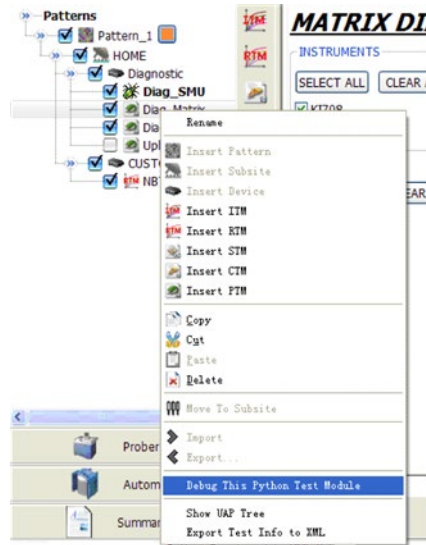
Figure 29: Enable Debug in View drop-down list



When you want to debug a PTM, or multiple PTMs, you can right-click the PTM module in the test tree. You will see a drop-down menu. Choose the Debug This Python Module option (see next Figure).

## NOTE

You must select each PTM, one at a time, if you want to debug multiple PTMs. Additionally, once you have completed debugging, you must deselect each PTM, one at a time, in order to return to normal operating mode in ACS.

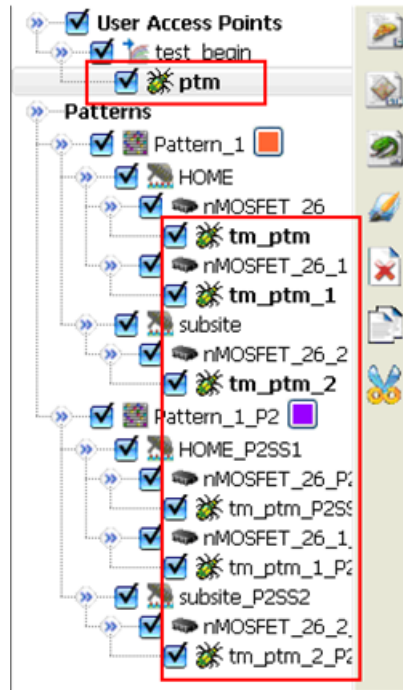




## NOTE

Once you select Debug This Python Test Module, the icon of the PTM will change and the text will change to bold. The icon will look like a bug for the selected PTM in the test tree (see next Figure).

**Figure 30: PTM selected for debugging**



**Start debugging**

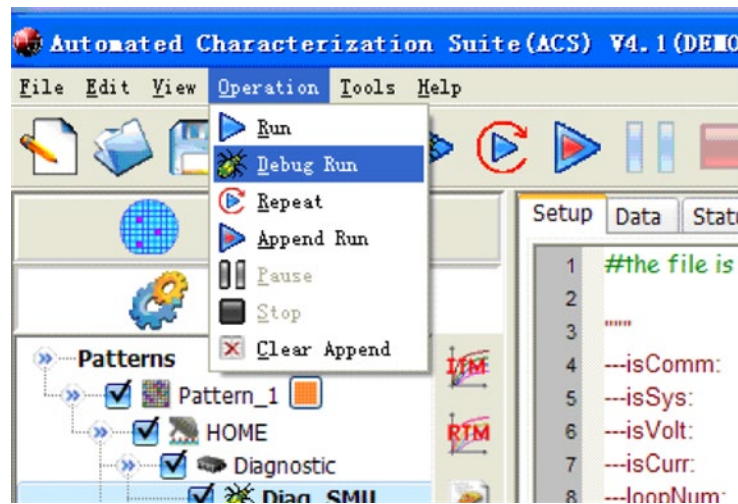
**NOTE**

Here are some notes for you to keep in mind before you start debugging modules in ACS:  
 You can click different levels to highlight in the test tree or in the user access points (UAPs).

- Click the PTM to debug individual module.
- Click the device to debug all modules under the device, one by one.
- Click the subsite to debug all modules under the subsite, one by one.
- Click the pattern to debug all modules under the subsite, one by one.
- In the UAP, click the PTM to debug individual UAP.
- In the test tree with added UAPs, click the modules in the test tree to test and the related UAPs will also be debugged; the UAPs include, test\_begin, test\_end, device\_begin, device\_end, subsite\_begin, subsite\_end, pattern\_begin, and pattern\_end.

1. Click the Operation function in the main toolbar.
2. In the drop-down menu, select Debug Run to start debugging the selected PTM (see next Figure):

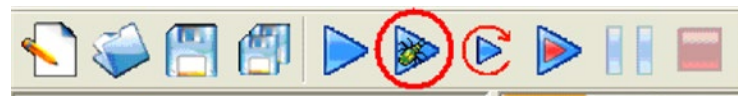
**Figure 31: Start Debug Run feature**



**NOTE**

You can also use the ACS toolbar to accomplish the debugging task by selecting the icon. The Debug Run icon is circled in the figure below (see next Figure):

**Figure 32: Debug icon in toolbar**

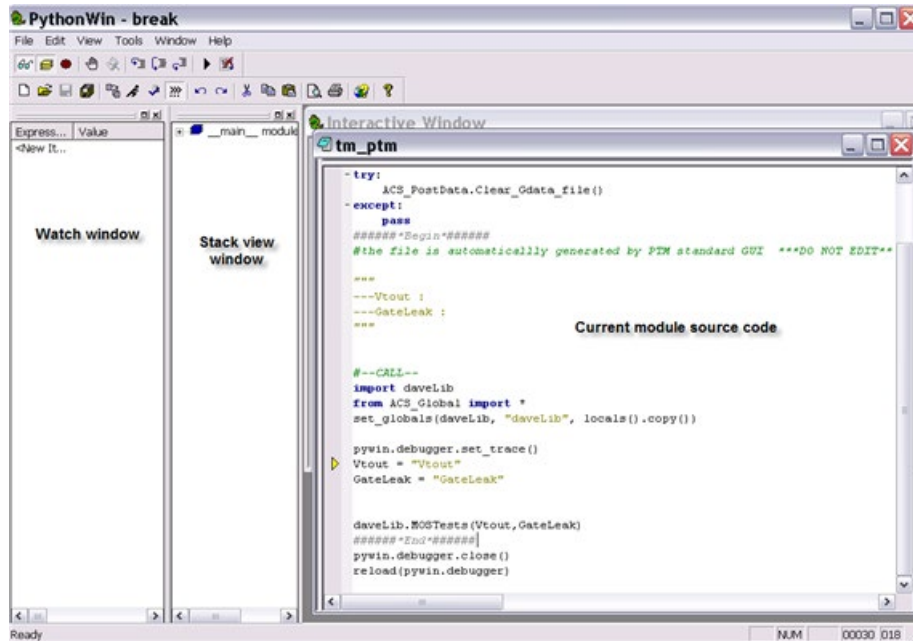


View the results of the debugging in the PythonWin - break window (see next Figure).

## NOTE

Once you select Debug Run, a new dialog box opens. This is where you get the results of debugging and where you will see the Watch window, Stack view window, and the current module source code. The Watch window and Stack view window are dockable, which means they can be moved for your viewing convenience.

Figure 33: PythonWin - break window



When debugging, if there are data output, it will be printed on the interactive window.

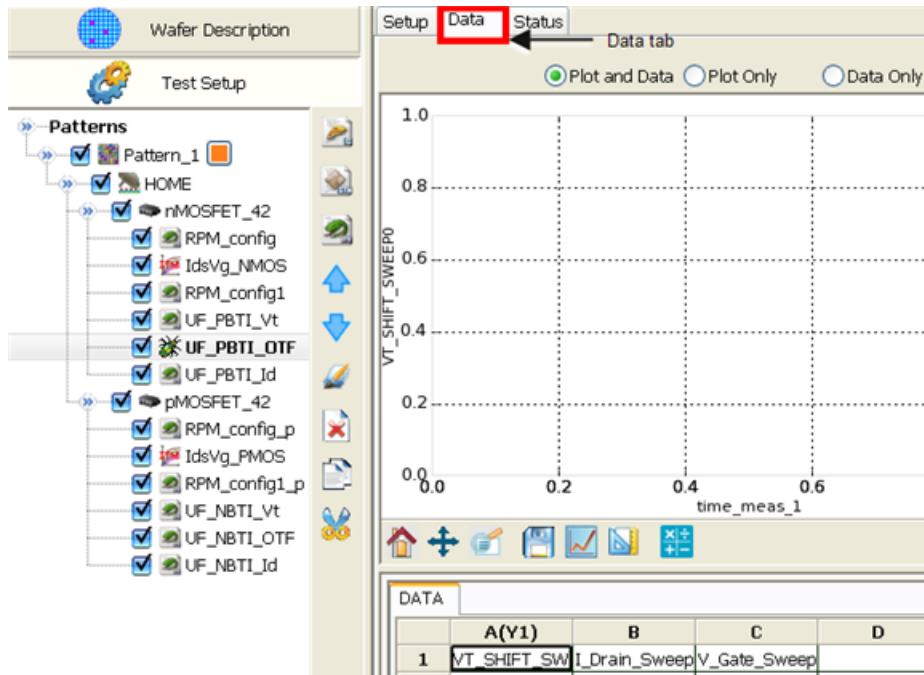
When the debugging is complete and you have changed the script, click the Save or Save All function on the debugging window toolbar to save it. The script will be saved in the default folder path: C:\ACS\KATS\Debug. The script file name will be the same as the module name.

NOTE

If you want to run debug on a saved file, you will need import the file from the folder C:\ACS\KATS\Debug before running debug.

You can close the debugging tool by clicking the X and closing the window. If there was data output from the debugging session, it will post to the ACS Data tab of the module (see next Figure).

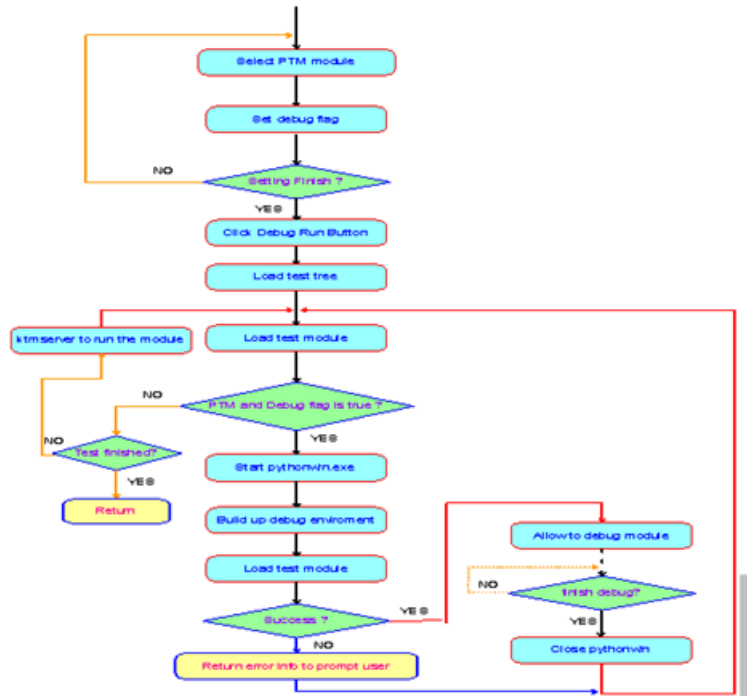
**Figure 34: PTM debugging Data tab**



**PTM debug flow chart**

Review the debug flow chart for a detailed list of step-by-step instructions (see next Figure):

**Figure 35: Debug flow chart**



**Pythonwin Debugging Toolbar icons and functions:**

**View Watch window**

Click the Watch window icon in order to create and monitor variables that you insert in the watch window (see next Figure).

NOTE

You can click the following icons to view or hide the windows: Watch, Stack view, and the Breakpoint list.

**Figure 36: Watch window icon**

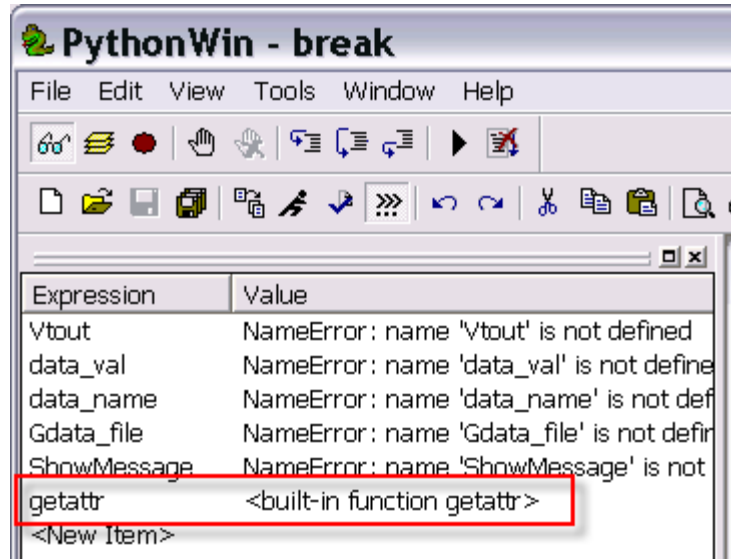


Create variables in the Watch window to closely monitor the source code and when stepping into, stepping over, or stepping out of the module source code.

NOTE

In the next Figure, note that the Expression (the variables) are listed in the order they are created. Plus, the variables that are in the active window will display with the current information (see the next Figures).

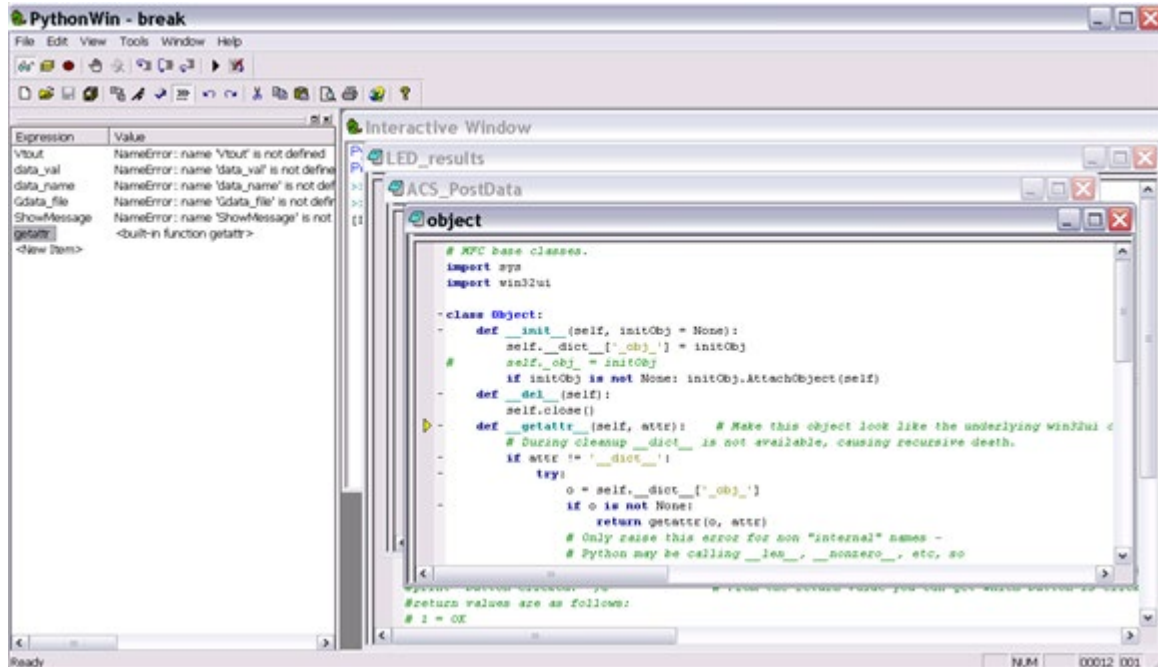
**Figure 37: Watch window variables**



## NOTE

In the next Figure, note that the Expression (the variable) named "getattr" in displaying information and that the active window is "object." Also, note that the other variables have a Value stating NameError. This is because the variables are not in the active window. are listed in the order they are created. Plus, the variables that are in the active window will display with the current information (see the next Figures).

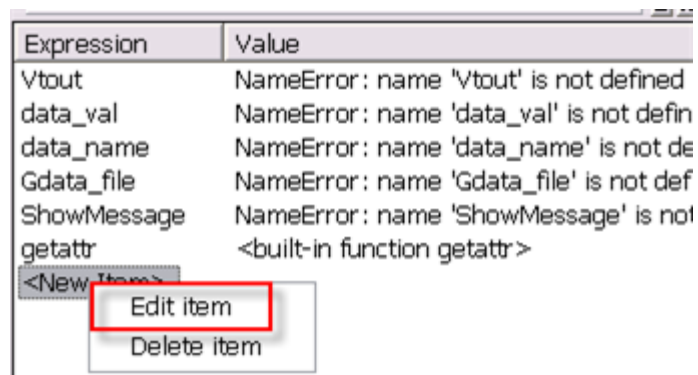
**Figure 38: Watch window variables active window**



### Create variables in the Watch window

1. Click the Expression <New Item>.
2. Right-click <New Item>.
3. Choose Edit Item in the drop-down box.
4. Type the name of a variable that you want to monitor (see next Figure).

**Figure 39: Create a variable in the Watch window**



### View the Stack view window

Click the Stack view icon (see next Figure).

Figure 40: Stack view icon

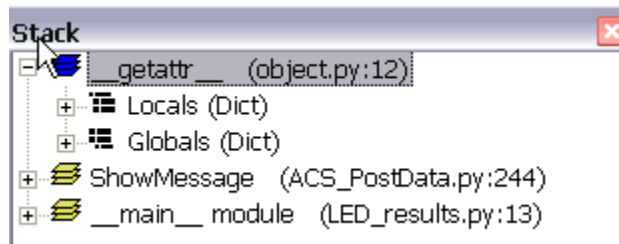


The Stack view window appears (see next Figure). In the Stack view you will see the variables that are active in the Watch window. You can also local and global variables by expanding the top level variable (see next Figure).

## NOTE

When the Stack view window opens, it may be docked to the PythonWin window, or it may be free floating (undocked)(see next Figure). If it is undocked, you will see the heading Stack on the window. If you move it to the Watch window it will dock side-by-side.

Figure 41: Stack view window



### View the Breakpoint list and Toggle Breakpoints

Click the Breakpoint list icon (see next Figure).

Figure 42: Breakpoint list icon



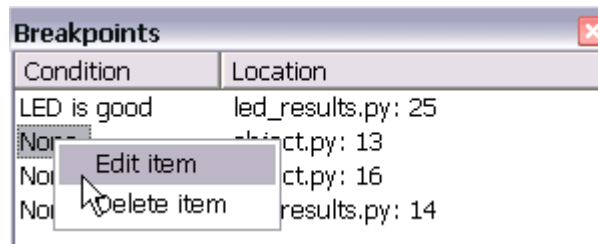


The Breakpoint list window appears (see next Figure). In the Breakpoint list window you will see the location of the breakpoints that you have created. You will also see the line number. For instance, in the next Figure, the location of `led_results.py:25` is line 25.

## NOTE

You will see a column named Condition in the Breakpoints window. For this function, you can click the word "None," right-click and choose Edit item. From there, you choose the name of the condition for the breakpoint that you configured in the module source code. Additionally, you can delete the breakpoint by using this method.

**Figure 43: Breakpoint list window**



When you step into a module's code, you can toggle breakpoints. The breakpoints are added by clicking on the Toggle Breakpoint icon (see next Figure).

**Figure 44: Toggle Breakpoint icon**

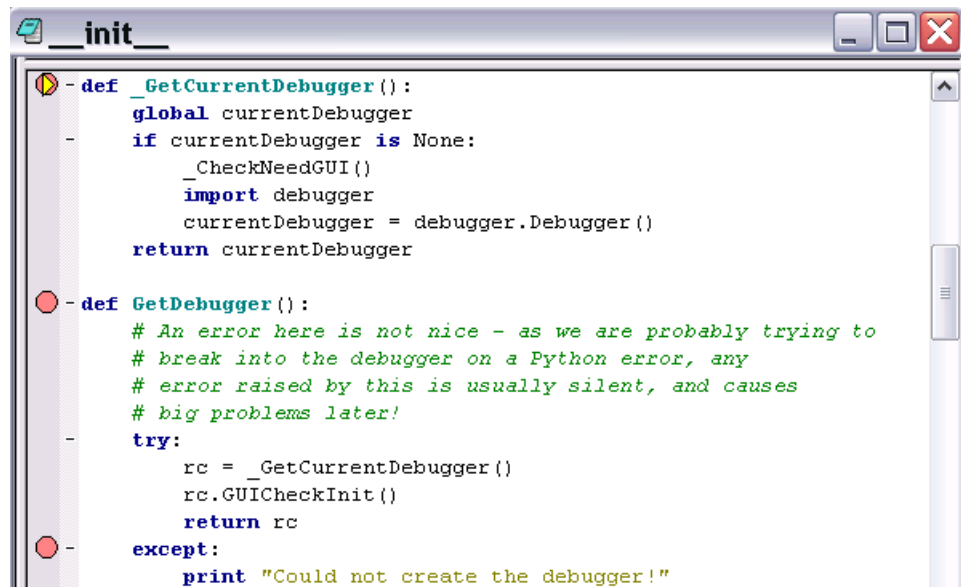


The breakpoints are displayed to the left of the code as a red dot (see next Figure).

## NOTE

You can create breakpoints by moving the cursor in the module source code to the line where a breakpoint is needed. Once the cursor is at the appropriate line, click the Toggle Breakpoint icon to add a breakpoint.

Figure 45: Breakpoint icons in module source code



### Clear All Breakpoints

When you want to delete all of the breakpoints, you must click the Clear All Breakpoints icon (see next Figure).

## NOTE

When you clear all of the breakpoints, the Breakpoints list window should not contain any breakpoints.

Figure 46: Clear All Breakpoints icon



### Step, Step over, Step out, and Go functions

When you step into the current module source code and execute debugging, line-by-line, you must click the Step icon (see next Figure).

**Figure 47: The Step icon**



When you step over, or skip, the current module source code and execute debugging at the next line, you must click the Step over icon (see next Figure).

**Figure 48: The Step over icon**



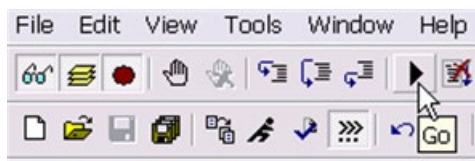
When you step out of the current module source code and execute debugging, you must click the Step out icon (see next Figure).

**Figure 49: The Step out icon**



When you need to begin or continue executing debugging, you must click the Go icon (see next Figure).

**Figure 50: The Go icon**



### Close the debugger

When want to close, or stop, your session of debugging the active module code window, you must click the Close icon (see next Figure).

**Figure 51: The Close icon**



**Pythonwin Standard Toolbar icons and functions**

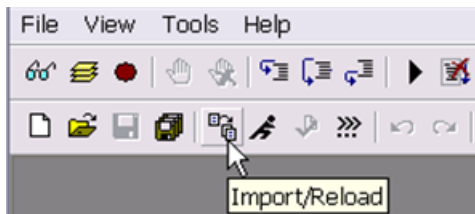
**Figure 52: Four specific PythonWin Standard Toolbar functions**



**Import/Reload a python script**

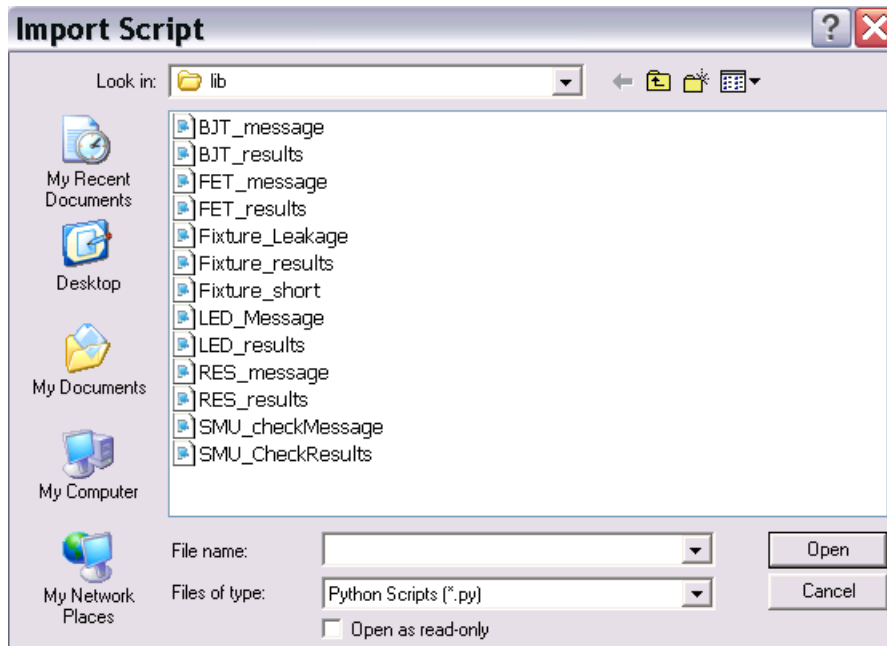
When want to import or reload python scripts, you must click the Import/Reload icon (see next Figure).

**Figure 53: The Import\_Reload icon**



When you click the Import/Reload icon, you will get a dialog box where you have to find the python script that you need (see next Figure). Once you open the script, you can begin your debugging session. Also, you can check the Interactive Window to find the condition and location of the module that plan to debug.

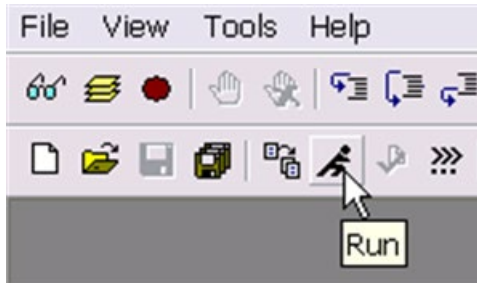
**Figure 54: The Import\_Reload dialog box**



**Run a script**

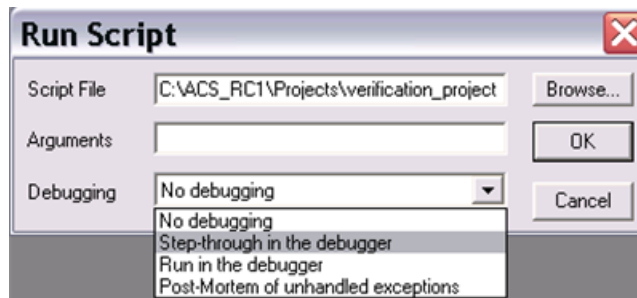
When want to run a python script, after importing or reloading, you must click the Run icon (see next Figure).

**Figure 55: The Run icon**



When you click the Run icon, you will get a dialog box. First, you must locate (browse) the script file. Second, you must enter any arguments. These are parameters that you establish to the function that you want to use while running the script. Third, you must click the down-arrow to choose the type of debugging desired when running a script. There are four choices: No debugging, Step-through in the debugger, Run in the debugger, Post-Mortem of unhandled exceptions (see next Figure).

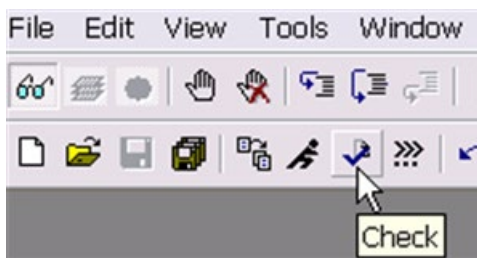
**Figure 56: PythonWin Run Script**



**Check a script**

When want to check a python script, without executing it, to make sure that it is a valid script for debugging, you must click the Check icon (see next Figure).

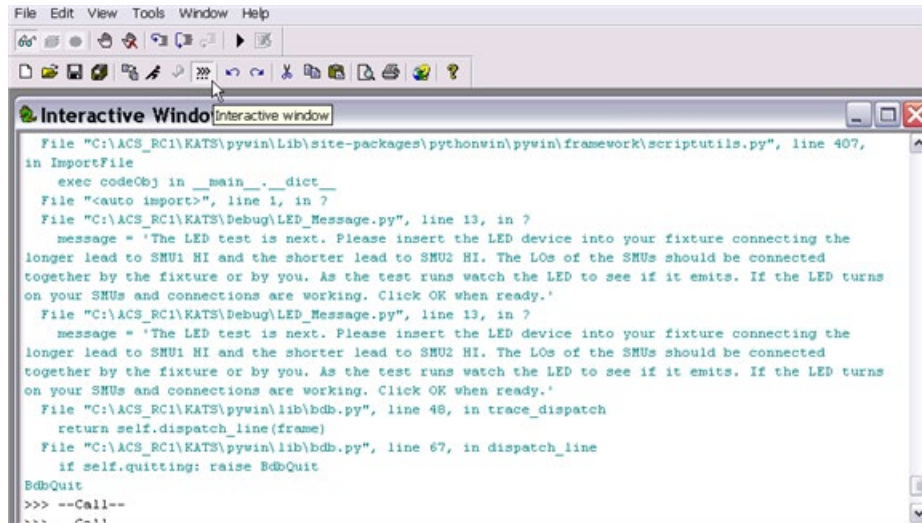
**Figure 57: The Check icon**



## Interactive Window

When want to check the status of your debugging session, the Interactive Window contains all of the command lines for the currently active python module window. You can view or hide this window by clicking the icon (see next Figure).

Figure 58: Interactive Window



```
File "C:\ACS_BC1\KATS\pywin\Lib\site-packages\pythonwin\pywin\framework\scriptutils.py", line 407,
in ImportFile
  exec codeObj in __main__.__dict__
File "<auto import>", line 1, in ?
File "C:\ACS_BC1\KATS\Debug\LED_Message.py", line 13, in ?
  message = 'The LED test is next. Please insert the LED device into your fixture connecting the
longer lead to SMU1 HI and the shorter lead to SMU2 HI. The LOs of the SMUs should be connected
together by the fixture or by you. As the test runs watch the LED to see if it emits. If the LED turns
on your SMUs and connections are working. Click OK when ready.'
File "C:\ACS_BC1\KATS\Debug\LED_Message.py", line 13, in ?
  message = 'The LED test is next. Please insert the LED device into your fixture connecting the
longer lead to SMU1 HI and the shorter lead to SMU2 HI. The LOs of the SMUs should be connected
together by the fixture or by you. As the test runs watch the LED to see if it emits. If the LED turns
on your SMUs and connections are working. Click OK when ready.'
File "C:\ACS_BC1\KATS\pywin\lib\bdb.py", line 46, in trace_dispatch
  return self.dispatch_line(frame)
File "C:\ACS_BC1\KATS\pywin\lib\bdb.py", line 67, in dispatch_line
  if self.quitting: raise BdbQuit
BdbQuit
>>> --Call--
... Call
```

## Debug tool limitations

### PythonWin limitations

ACS integrates third-party tools to achieve the debug function and therefore will not prevent you from opening other modules or using other features. If you directly open a python test module from PythonWin and debug it, you may get inaccurate, inconsistent, or strange results that you would not normally receive. Therefore, it is suggested that you use the debug tool as intended.

### Debug hardware limitations

You can control instruments, such as the KI26xx, KI4200, KI2400, etc. with a PTM. Additionally, these instruments can be connected differently and the type of connection requires different debugging environments:

- GPIB
- KXCI
- LXI

### GPIB control

Most instruments can be connected by GPIB, such as the series KI26xx, series KI24xx, series KI3700, series KI24xx, etc. The PythonWin debug tool for ACS software supports debug testing instruments that are connected by GPIB.

### KXCI control

ACS controls the model KI4200 using the KXCI interface. ACS is able to send commands to KXCI and KXCI will parse the commands and control hardware according to the command. ACS can be installed on a PC or on the Model 4200-SCS.

#### A. Installing ACS on the Model 4200-SCS

- The debug tool supports the debug testing on the PTM with the Model 4200-SCS.

#### A. Installing ACS on a PC

- The debug tool only supports the PTM that directly sends KXCI commands to the Model KI4200.
- If the PTM includes a LPT module, the PTM cannot control hardware, therefore, the debug tool is not supported for debugging the module.

### LXI control

Since the LXI mode only supports ITM and STM, the debug tool is not supported for debugging PTMs if the hardware is connect using LXI.

### Other limitations:

#### Step in mode

If you import modules to your PTM that contain the following file suffix (.pyc) or the modules have been imported from a zipped library, such as kimisc, ACS\_PostData, ACSLPT, etc, you will not be able to use the Step in command to enter into the module and single step through the source code.

#### PTM auto update

You can modify a debugged PTM during the process of debugging, however, any changes you make to the debugged PTM are not automatically updated to the original PTM in your test. Therefore, it is necessary to manually make the same changes to your PTM in the ACS file to ensure that the original matches the debugged PTM.

## Series 2600B Library and Python Library

### In this section:

Series 2600B library introduction .....	4-1
Device library .....	4-4
WLR library overview .....	4-24
Python user library introduction.....	4-53

### Series 2600B library introduction

The ACS software has a test library of commands (26Library), that is a library for the Series 2600B SourceMeter® instruments. The library can be found 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 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_ICBO	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_ivevb	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_lfd_Vfd
Diode_lfd_Vfd_vsweep	Diode_Ileakage_Vrd	Diode_Ird_Vrd_vsweep
Diode_Vrd_Ird	Diode_Vbr_Ird	Diode_Vfd_lfd



<b>Resistor Test Library</b>	
Resistor_single	Resistor_sweep
VDP_Resistivity	

The next table summarizes all of the test modules that are in the WLR library. More detailed descriptions of each module follow the tables.

<b>Wafer-level reliability (WLR) Test Library</b>		
HCI	NBTI	NBTI_on_the_fly
TDDB_CCS	TDDB_per_pin	qbd_rmpj
qbd_rmpv	Em_iso_test	NBTI_meas

## Create a library without Script Editor

You can use Test Script Language on the Keithley Instruments Series 2600B System SourceMeter or the Linear Parametric Test Library (LPT Library) to write a new library.

### NOTE

You will need to use the syntax of the script that follows the rules for LUA programming language.

The script must use the `postdata`, `postbuffer`, or the `posttable` function to retrieve data from the Series 2600B. Refer to the LPT Library Reference in the Series 2600B Reference manual for more information on these functions. For examples, refer to the directory: `\\ACS \Library\26Library` folder.

If you would like to design a test library with a graphical user interface (GUI), follow the instructions below:

- The first line must be the name of the `.xrc` (GUI) file, and the `.xrc` (GUI) file must be put in the `\\ACS\library\26Library\xrc` folder. ACS will then load the GUI file automatically when importing the script file. For example:
  - `----<xrc=HCl.xrc>----`
- The types of input variables must be:
  - `instid` (SMU input)
  - `string`
  - `double`
  - `integer`
  - `table`
- You can set a default value for every input variable. You can also set the input range for double and integer-type input variables. For example:
  - `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, Vrd <= 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`
- 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--`. For example:
  - `-- 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' will reset instruments after test, '0' will not.`
  - `-- END OF INPUT --`
- 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` folder.

## Create a library using Script Editor

You can also use the Script Editor in ACS to write a new library. For more information about how to use the Script Editor, refer to the Script Editor Tool.

When you use the Script Editor, the library will be automatically saved to \\ACS\library\26Library\TSPLib. The .xrc GUI will be automatically saved to \\ACS\library\26Library\TSPLib\xrc.

## Device library

### BJT library overview

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

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

This BJT test library is used to test parameters of a BJT, such as breakdown voltage, amplify times, reverse current, gummel plot, etc. The 26library is used with a Series 2600B instrument to create test script files, based on the Series 2600B LPT library. The 42library is used with a Model 4200 to create KULT files, based on the Model 4200 LPT library.

#### Three\_term\_BJT\_BVCBO

---

Module Name: Three\_term\_BJT\_BVCBO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-base breakdown voltage of the BJT, with the emitter open.

Pin Connections: Open the emitter and apply the desired current to the collector. The base connects to ground.

Intended results: Get the collector-base breakdown voltage.

#### Three\_term\_BJT\_BVCEI

---

Module Name: Three\_term\_BJT\_BVCEI

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter breakdown voltage of the BJT with a biased current forced at the base.

Pin Connections: Apply the desired 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**

---

Module Name: Three\_term\_BJT\_BVCEO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter breakdown voltage with the base open.

Pin Connections: Open the base and apply the desired current to the collector. The emitter connects to ground.

Intended results: Get the collector-emitter breakdown voltage.

### **Three\_term\_BJT\_BVCES**

---

Module Name: Three\_term\_BJT\_BVCES

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter breakdown voltage of the BJT with a short at the base-emitter.

Pin Connections: Apply the desired current to the collector. The base and emitter connect to ground.

Intended results: Get the collector-emitter breakdown voltage.

### **Three\_term\_BJT\_BVCEV**

---

Module Name: Three\_term\_BJT\_BVCEV

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter breakdown voltage with a biased voltage forced at the base.

Pin Connections: Apply the desired 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**

---

Module Name: Three\_term\_BJT\_BVEBO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the emitter-base breakdown voltage of the BJT with the collector open.

Pin Connections: Open the collector and set the emitter to the desired current. Connect the base to ground.

Intended results: Get the emitter-base breakdown voltage.

### **Three\_term\_BJT\_BVECO**

---

Module Name: Three\_term\_BJT\_BVECO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the emitter-collector breakdown voltage of the BJT with the base open.

Pin Connections: Open the base and apply the desired current to the emitter. Connect the collector to ground.

Intended results: Get the emitter-collector breakdown voltage.

### **Three\_term\_BJT\_HFE\_sw**

---

Module Name: Three\_term\_BJT\_HFE\_sw

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the DC current gain of the BJT with a collector voltage sweep.

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's not connected to ground be sure to apply a bias voltage on the emitter).

Use this technique:

1. Force collectorV sweep.
2. Measure  $I_b$  and  $I_c$ .
3. Check for measurement problems.
4. Calculate  $HFE = I_c/I_b$ .

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

### Three\_term\_BJT\_HFE\_trial

---

Module name: Three\_term\_HFE\_trial

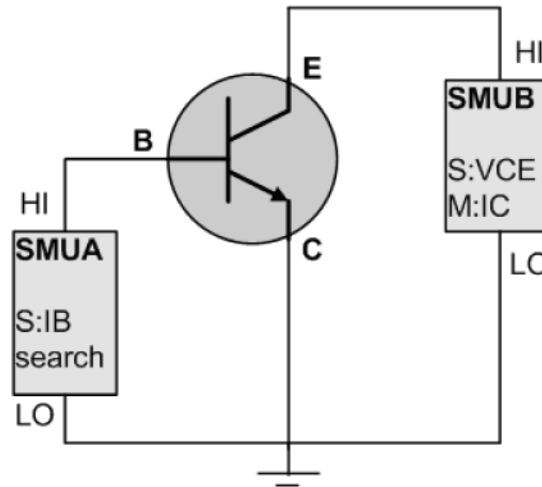
Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the DC current gain of the BJT with a traditional test method.

Pin Connections: Sharing the emitter connection, apply a desired 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; see next Figure).

**Figure 59: Three\_term\_BJT\_HFE\_trial pin connection**



Use this technique:

1. Set base current level, measure collector current, and check if it equals the desired target.
2. If collector current is not at desired target, repeat step 1.
3. Find the target collector current and record base current.
4. Calculate  $HFE = I_{C\_tar} / I_B$ .

Intended Results: Get the the DC current gain.

### Three\_term\_BJT\_IBCO

---

Module Name: Three\_term\_BJT\_IBCO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the base-collector current with the emitter open.

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**

---

Module Name: Three\_term\_BJT\_IBEO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the base-emitter current with the collector open.

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 desired bias voltage.

Intended results: Get the base-emitter current.

### **Three\_term\_BJT\_ibicvbe**

---

Module Name: Three\_term\_BJT\_ibicvbe

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the base current and collector current of the BJT with a specified base voltage sweep.

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 the desired bias voltage.

Use this technique:

1. Measure base current and collector current of BJT.
2. Get  $I_b$ - $V_{be}$  and  $I_c$ - $V_{be}$  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**

---

Module Name: Three\_term\_BJT\_ibvbe

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the base current of the BJT with a specified base voltage sweep.

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 desired bias voltage.

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

---

**Three\_term\_BJT\_ICBO**

---

Module Name: Three\_term\_BJT\_ICBO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT.

Function: Test the collector-base cut off current with the emitter open.

Pin Connections: Open the emitter and apply a desired voltage to the collector. The base is connected to ground.

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

---

**Three\_term\_BJT\_ICEO**

---

Module Name: Three\_term\_BJT\_ICEO

Instrument: Keithley Instruments Model 4200.

DUT: Three-terminal BJT

Function: Test the collector-emitter cut off current with the base open.

Pin Connections: Open the base, apply a desired voltage to the collector, and connect the emitter to ground.

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

---

**Three\_term\_BJT\_ICES**

---

Module Name: Three\_term\_BJT\_ICES

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter cut-off current with the base-emitter shorted.

Pin Connections: Apply a desired voltage to the collector and connect the emitter and base to ground.

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

---

**Three\_term\_BJT\_ICEV**

---

Module Name: Three\_term\_BJT\_ICEV

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter cut off current with a bias voltage on the base.

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

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



---

### **Three\_term\_BJT\_icvcb**

---

Module Name: Three\_term\_BJT\_icvcb

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-current of the BJT with a specified collector voltage sweep.

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 the desired bias voltage.

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

---

### **Three\_term\_BJT\_icvce\_biasIB**

---

Module Name: Three\_term\_BJT\_icvce\_biasIB

Instrument: Keithley Instruments Model 4200

DUT: Three-terminal BJT

Function: Test a series of collector-current and collector-emitter voltage ( $I_c$ - $V_{ce}$ ) curves of the BJT while stepping the base current.

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**

---

Module Name: Three\_term\_BJT\_icvce\_biasVB

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test a series of  $I_c$ - $V_{ce}$  curves of the BJT while stepping the base voltage.

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**

---

Module Name: Three\_term\_BJT\_icvce\_stepib

Instrument: Keithley Instruments Model 4200

DUT: Three-terminal BJT

Function: Test a series of  $I_c$ - $V_{ce}$  curves of the BJT while stepping the base current.

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**

---

Module Name: Three\_term\_BJT\_icvce\_stepvb

Instrument: Keithley Instruments Model 4200

DUT: Three-terminal BJT

Function: Test a series of Ic-Vce curves of the BJT while stepping the base voltage.

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**

---

Module Name: Three\_term\_BJT\_IEBO

Instrument: Keithley Instruments Model 4200 .

DUT: Three-terminal BJT

Function: Test the emitter-base cut off current with the collector open.

Pin Connections: Open the collector and apply a desired voltage to the emitter. The base connects to ground.

Intended results: Get the emitter-base cut-off current.

---

**Three\_term\_BJT\_IECO**

---

Module Name: Three\_term\_BJT\_IECO

Instrument: Keithley Instruments Model 4200

DUT: Three-terminal BJT

Function: Test the emitter-collector current with the base open.

Pin Connections: Open the base and apply a desired 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**

---

Module Name: Three\_term\_BJT\_ieveb

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the emitter-current of the BJT with a specified emitter voltage sweep.

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

---

Module Name: Three\_term\_BJT\_VBCO

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the base-collector voltage of the BJT with the emitter open.

Pin Connections: Open the emitter and apply a current to the base. The emitter usually connects to ground, but can be set to the desired bias voltage.

Intended results: Get the base-collector voltage.

---

### Three\_term\_BJT\_VCE

---

Module Name: Three\_term\_BJT\_VCE

Instrument: Keithley Instruments Series 2600B

DUT: Three-terminal BJT

Function: Test the collector-emitter voltage BJT.

Pin Connections: Apply voltage on the base, set the collector-current to a desired level, and connect the emitter to ground.

Intended results: Get the collector-emitter voltage.

## MOSFET library overview

The MOSFET library components are located in the following directories:

- \\ACS \Library\26Library\Parametric\MOSFET
- \\ACS \Library\42Library\Parametric\S4200ParLib\src

The MOSFET parameter library is used to test parameters of a MOSFET, such as  $g_{m\text{lin}}$ ,  $g_{m\text{sat}}$ ,  $i_{d\text{vd}}$ ,  $i_{d\text{vg}}$ ,  $i_{g\text{vg}}$ ,  $V_{t\text{ci}}$ , and  $V_{t\text{ex}}$ . The 26library is used with a Series 2600B to create test script files, based on the Series 2600B LPT library. The 42library is used with a Model 4200 to create KULT files, based on the Model 4200 LPT library.

---

### Four\_term\_MOSFET\_BVDSS

---

Module Name: Four\_term\_MOSFET\_BVDSS

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-source breakdown voltage of a MOSFET with the gate-source shorted.

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 voltage. The gate and source are connected to ground, if they are not connected to ground, force 0 voltage.

Intended results: Get the breakdown voltage between the drain and source with gate-source shorted.

---

**Four\_term\_MOSFET\_BVDSV**

---

Module Name: Four\_term\_MOSFET\_BVDSV

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-source breakdown voltage of a MOSFET with the gate biased.

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**

---

Module Name: Four\_term\_MOSFET\_BVGSO

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the gate-source breakdown voltage of a MOSFET with the drain opened.

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**

---

Module Name: Four\_term\_MOSFET\_BVGDS

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the gate-drain breakdown voltage of a MOSFET with the source-drain shorted.

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**

---

Module Name: Four\_term\_MOSFET\_BVGDO

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the gate-drain breakdown voltage of a MOSFET with the source opened.

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**

---

Module Name: Four\_term\_MOSFET\_IDL

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to measure the drain leakage-current with the gate-source shorted.

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**

---

Module Name: Four\_term\_MOSFET\_IDS\_ISD

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to measure the drain-source and the source drain-current with the gate biased.

Pin Connections: Apply voltage separately on the gate, source, and drain. The bulk is usually connected to ground, but can be set with a desired bias voltage.

Intended results: Get the measured drain-source and source drain-current with the gate biased.

---

**Four\_term\_MOSFET\_idvd**

---

Module Name: Four\_term\_MOSFET\_idvd

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-current at a specified drain-voltage sweep.

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**

---

Module Name: Four\_term\_MOSFET\_idvd\_vg

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test a series of  $I_d$ - $V_d$  curves for a four-terminal MOSFET, that performs on the Series 2600B.

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**

---

Module Name: Four\_term\_MOSFET\_idvg

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-current at a specified gate-voltage sweep.

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**

---

Module Name: Four\_term\_MOSFET\_idvg\_vd

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-current at a specified gate-voltage sweep while stepping the drain.

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**

---

Module Name: Four\_term\_MOSFET\_idvg\_vsub

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the drain-current at a specified gate-voltage sweep while stepping the bulk.

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**

---

Module Name: Four\_term\_MOSFET\_IGL

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to measure the gate leakage-current while the source-drain shorted.

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**

---

Module Name: Four\_term\_MOSFET\_igvg

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the gate-current at a specified gate-voltage sweep when the drain is biased.

Pin Connections: Bias the drain and sweep the gate. Connect the bulk and source to ground, otherwise, apply the desired voltage.

Intended results: Get the measured gate-current at the gate-voltage sweep.

### **Four\_term\_MOSFET\_ISL**

---

Module Name: Four\_term\_MOSFET\_ISL

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to measure the source leakage-current when gate-drain is shorted.

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\_ishvg**

---

Module Name: Four\_term\_MOSFET\_ishvg

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to test the bulk-current at a specified gate-voltage sweep.

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

---

Module Name: Four\_term\_MOSFET\_Vth\_ci

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to get the constant current threshold voltage of a 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 below:

$V_{th\_ci} = V_{GS} (@ID = 1 \mu A.W/L)$  -- NMOS

$V_{th\_ci} = V_{GS} (@ID = -0.025 \mu A.W/L)$  -- PMOS

Where W and L are the gate-width and gate-length as printed on the wafer. Set a target drain-current  $I_{d\_tar}$  ( $I_{d\_tar} = 1 \mu A.W/L$ , or  $-0.025 \mu A.W/L$ ), which means it is near the threshold, then search the gate-voltage to make the drain-current equal to  $I_{d\_tar}$ .

### NOTE

The Four\_term\_MOSFET\_Vth\_ci measurement technique must determine  $V_{th\_ci}$  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  $V_{th\_ci}$  measurements are  $V_{DS} = V_{DS\_lin}$ ,  $V_{BS} = V_{BB}$  for linear region measurement, or  $V_{DS} = V_{DS\_sat}$ , ( $V_{BS} = V_{BB}$  for saturation region measurement). Typically, for PMOS,  $V_{DS\_lin} = -0.1 V (@VDD = 5V)$ ; for NMOS,  $V_{DS\_lin} = 0.1V (@VDD = 5V)$ .

Intended results: Get the constant-current threshold voltage.



## Four\_term\_MOSFET\_Vth\_ex

---

Module Name: Four\_term\_MOSFET\_Vth\_ex

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to get threshold voltage from the maximum slope measurement.

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:

$V_{th\_ex} = V_{GS} (@G_{mmax}) - ID(@G_{mmax}) / G_{mmax}$

Where  $V_{GS} (@G_{mmax})$  is the gate-voltage at the point of the maximum slope of the ID-VGS curve and  $ID(@G_{mmax})$  is the drain-current at the point of the maximum slope of the ID-VGS curve. Also,  $G_{mmax}$  is the maximum slope of the ID-VGS curve.

### NOTE

DC bias voltages for  $V_{th\_ex}$  measurements are  $V_{DS} = V_{DS\_lin}$ ,  $V_{BS} = V_{BB}$  for linear measurement.

$V_{DS} = V_{DS\_sat}$ ,  $V_{BS} = V_{BB}$  for saturation. Typically, for PMOS,  $V_{DS\_lin} = -0.1 V (@V_{DD} = 5V)$ ; for NMOS,  $V_{DS\_lin} = 0.1V (@V_{DD} = 5V)$ .

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 ( $V_{th\_ex}$ ).

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\_Ilsq

---

Module Name: Four\_term\_MOSFET\_Vth\_Ilsq

Instrument: Keithley Instruments Series 2600B

DUT: Four-terminal MOSFET

Function: Used to extract the threshold voltage from the measurement slope. In this test, the least-square approximation is used.

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:

$V_{th\_ex} = V_{GS}(@G_{mmax}) - ID(@G_{mmax}) / G_{mmax}$ .

Where  $V_{GS}(@G_{mmax})$  is the gate-voltage at the point of the maximum slope of the ID-VGS curve and  $ID(@G_{mmax})$  is the drain-current at the point of the maximum slope of the ID-VGS curve. Also,  $G_{mmax}$  is the maximum slope of the ID-VGS curve.

### NOTE

DC bias voltages for  $V_{th\_ex}$  measurements are  $V_{DS} = V_{DS\_lin}$ ,  $V_{BS} = V_{BB}$  for linear measurement.

$V_{DS} = V_{DS\_sat}$ ,  $V_{BS} = V_{BB}$  for saturation.

Typically, for PMOS,  $V_{DS\_lin} = -0.1 V (@V_{DD} = 5V)$ ; for NMOS,  $V_{DS\_lin} = 0.1V (@V_{DD} = 5V)$ .

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 ( $V_{th\_ex}$ ).

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

Module Name: Four\_term\_MOSFET\_Vth\_sense

Instrument: Keithley Instruments Series 2600B in sense mode

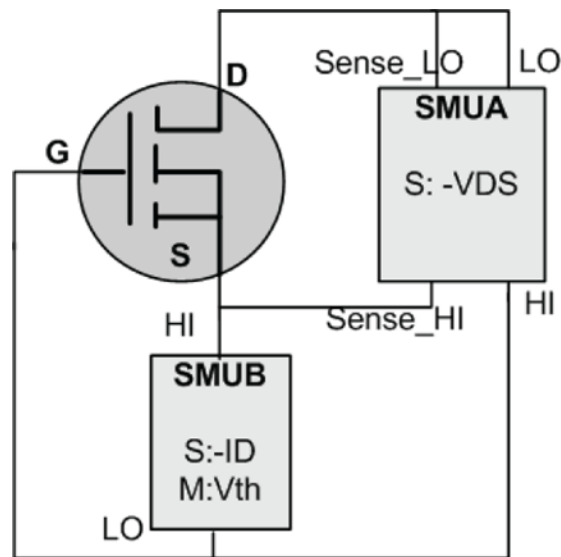
DUT: Four-terminal MOSFET

Function: Used to get the self-biasing threshold voltage ( $V_{th\_sel}$ ) for a 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 are together.

Use two SMUs: For example, SMU\_A and SMU\_B. SMU\_A is in sense mode.

**Figure 60: Four\_term\_MOSFET\_Vth\_sense pin connections**



Technique:

- Set SMU\_B to the desired drain-current level ( $I_{DS}$ ).
- Set SMU\_A to the desired VDS level.
- Measure the VG using SMU\_B.
- Ensure the threshold voltage equals the gate-voltage.

### NOTE

When in sense mode, SMU\_A will automatically vary the voltage on the gate until VDS is equal to the desired level, and at that time measure 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 located in the following directories:

- \\ACS \Library\26Library\Parametric\Diode
- \\ACS \Library\42Library\Parametric\S4200ParLib\src

The diode test library is used to test parameters of a diode, such as the forward voltage and current, reverse voltage and current, I-V curve, and dynamic impedance. The 26library is used with a Series 2600B to create test script files based on the Series 2600B LPT library. The 42library is used with a Model 4200 to create KULT files based on the Model 4200 LPT library.

---

### Diode\_DynamicZ

---

Module Name: Diode\_DynamicZ

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Calculates the dynamic impedance based on two forward voltage measurements or two reverse voltage measurements. For example,  $\text{DynamicZ} = (v_2 - v_1) / (I_2 - I_1)$ .

Pin Connections: Uses one SMU to force the forward current, while the other terminal is connected to ground.

Intended results: Get dynamic impedance.

---

### Diode\_Ild\_Vfd

---

Module Name: Diode\_Ild\_Vfd

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the the forward current of a diode at a specified forward voltage.

Pin Connections: Force a forward voltage on the P terminal. Connect the N terminal to ground.

Intended results: Get the forward current.

---

### Diode\_Ild\_Vfd\_vsweep

---

Module Name: Diode\_Ild\_Vfd\_vsweep

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the forward current with a forward voltage sweep in order to indicate the forward I-V characteristics of a 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**

---

Module Name: Diode\_Ileakage\_Vrd

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the leakage-current of a diode at a specified reverse voltage.

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**

---

Module Name: Diode\_Ird\_Vrd\_vsweep

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the reverse current with a reverse voltage sweep in order to indicate the reverse I-V characteristics of a 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**

---

Module Name: Diode\_Vbr\_Ird

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the breakdown voltage of a diode at a specified reverse current.

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\_lfd**

---

Module Name: Diode\_Vfd\_lfd

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the forward voltage of a 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**

---

Module Name: Diode\_Vrd\_Ird

Instrument: Keithley Instruments Series 2600B

DUT: Diode

Function: Test the reverse voltage of a diode at a specified reverse current.

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 located in the following directories:

- \\ACS \Library\26Library\Parametric\Resistor
- \\ACS \Library\42Library\Parametric\S4200ParLib\src

The resistor test library is used to test parameters of a resistor, such as resistance (source V measure I or source I measure V, 2-wire or 4-wire). The 26library is used with a Series 2600B to create test script files based on Series 2600B LPT library. The 42library is used with a Model 4200 to create KULT files based on the Model 4200 LPT library.

## **Resistor\_single**

---

Module Name: Resistor\_single

Instrument: Keithley Instruments Series 2600B

DUT: Two term generic device

Function: Measure Resistance at specified voltage or current stress.

Pin Connections: It supports 1 or 2 SMU(s), 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**

---

Module Name: Resistor\_sweep

Instrument: Keithley Instruments Series 2600B

DUT: Two term generic device

Function: Measure Resistance with specified voltage or current sweep.

Pin Connections: Supports 1 or 2 SMU(s). 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 wafer-level reliability (WLR) library components are located in the following directory:

- \\ACS\Library\26Library\WLR

The WLR test library provides certain wafer-level reliability tests on devices with Series 2600B instruments. The HCI, TDDDB, and two NBTI tests are available in this library. They are test script files, based on the Series 2600B LPT library.

## HCI

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: The 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 frame (set by you) is completed.

#### 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
```

#### Syntax:

```
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,]  --when t_mode is 0 is the time interval; when t_mode is
1 is the number of points in one decade

table time_input={0,1,2,5,10} -- when t_mode is 2 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 PLC 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 will be output; 0: The curve will not be 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 of 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)
double Id_target=1e-4 -- if not nil, Vtci will be 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)

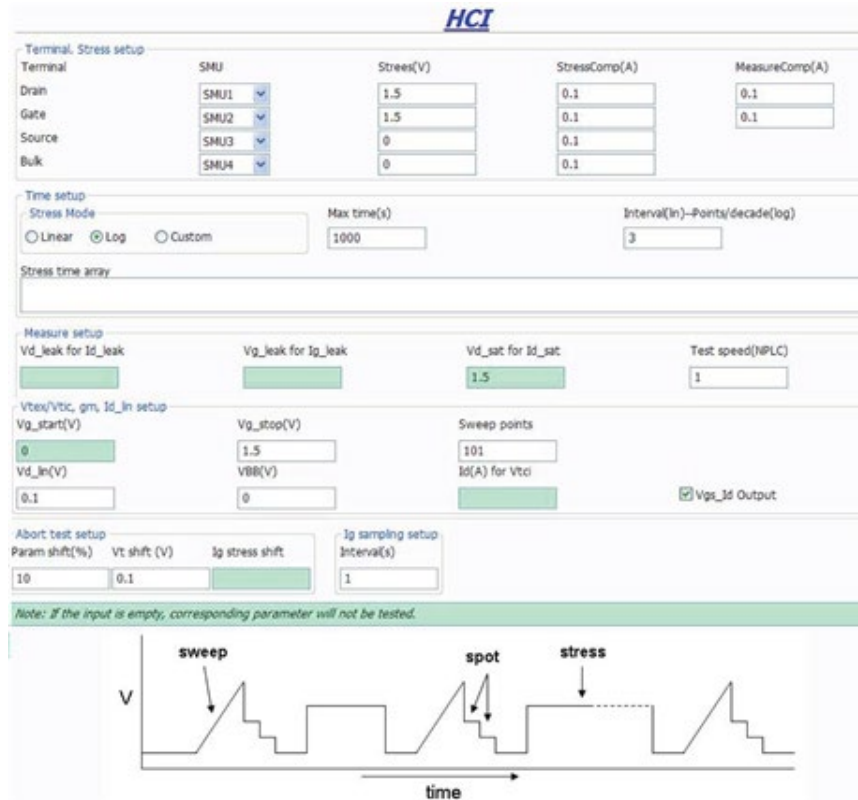
```

**GUI-Related**

The next Figure shows the GUI dialog box for HCI testing. If the test script processor (TSP™) file imported has a corresponding .xrc GUI file, ACS automatically loads and displays the GUI.

Refer to the .xrc GUI File for more information on importing .xrc files.

**Figure 61: GUI 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 and/or bulk are 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 will not be performed.

## NOTE

If the Vg\_start entry field is completed, but the Id(A) for Vtci entry field is empty, the threshold voltage (Vth) will be extracted from the maximum gm. If the Id(A) for Vtci entry field is also completed, the Vth will be 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 will be monitored during stress, and if Ig[now] = Ig stress shift\*Ig[fresh], the test ends.

### Example call:

```
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)
```

## TDDB\_CCS

This function is used to perform the constant current time-dependent dielectric breakdown (TDDB) test. Up to four SMUs are supported and only voltage is measured. The hard breakdown (HBD) occurs if:

- the  $V_g$  is below breakdown voltage ( $\text{abs}(V_g) < \text{abs}(V_{\text{min}})$ )
- the  $V_g$  falls dramatically ( $\text{abs}(V_g[\text{now}]) \leq \text{HBDL} * \text{abs}(V_g[\text{prev}])$ )

### Syntax:

```
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 BD.when  $V_g[\text{now}] \geq V_g[\text{prev}] * \text{HBDL}$  then
abort.
double V_min=0.06 in [0,200] --minimum voltage
double time_max=nil in(0, )/nil --max time of experiment. if 'nil' appears, test until BD
double holdtime=0 in[0, ) --time before stress begin (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 not 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 not 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 4 smus are supported. if not 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 not 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
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
```

The next Figure shows the dialog box for the TDDB\_CCS test. A general description of this dialog box is included below.

Figure 62: GUI for TDDB\_CCS

**TDDB ConstantCurrent**

SMU	Stress	Measure	Compliance
SMU1	1e-6	1	20
SMU2	1e-6	1	20
NONE			
NONE			

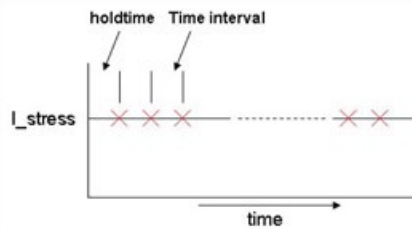
---

Hard BD limit:  V minimum:

---

Time Interval (s):  Time Max(s):  Holdtime(s):  NPLC:

*Note: If the input is empty, corresponding parameter will not be tested.*

**TDDB CCS GUI 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 zero if you only want to run a stress test.

Hard BD limit & 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 will continue until all devices fail.

**Example call:**

```

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).

```

## TDDB\_per\_pin

This function is used to perform a time-dependent dielectric breakdown (TDDB) test. Up to four SMUs are supported, voltage is forced, and current is measured.

1. If the breakdown mode is 0, hard breakdown (HBD) will be monitored. If the breakdown mode is 1, soft breakdown (SBD) will also be monitored.
2. HBD occurs when  $I_g[\text{now}] = \text{HBDL} * I_g[\text{prev}]$ .
3. To evaluate the SBD, calculate the noise of the gate-current ( $I_{noi}$ ) from the formula listed in JESD92. The base noise ( $I_{noi\_base}$ ) is calculated with the  $I_{noi}$  average value (AVL) and base number ( $bas\_num$ ). When  $I_{noi\_base}$  is set, SBD occurs if several sequential  $I_{noi}$ s conditions are met:  $I_{noi}[\text{now}] = \text{SBDL} * I_{noi\_base}$ .
4. If the DUT is a MOSFET, set the SMUs that do not need to measure to 0 ( $meas = 0$ ).
5. Intended outputs: time,  $I_g$ ,  $I_g\_noise$  (when SBD is required), and  $breakdown\_type$  of SMUs requiring measurement.

### Syntax:

```
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 BD. When  $I_g[\text{now}] >= I_g[\text{prev}] * \text{HBDL}$ , then abort.
integer BD_mode=0         --0: HBD only. All the parameters related to SBD could be set to
                           nil; 1: also SBD
double time_max=nil       --Max time of experiment; if 'nil' appears, test until BD
integer SBDL=500          --Limit of SBD; when  $I_{noi}[\text{now}] = I_{noi}[\text{base}] * \text{SBDL}$ , then abort
integer AVL=10            --Standard when calculating base noise current; if
                            $I_{noi}[\text{now}] <= \text{AVL} * I_{noi\_base}$ ,  $I_{noi}[\text{now}]$  should be included into
                            $I_{noi\_base}$  calculation
double holdtime=0         --Time before stress begins (Unit: S)
integer bas_num=6         --Number of noise current used to calculate  $I_{noi\_base}$  value
integer SBD_num=5        --Number of noise used to determine SBD
integer smu1=1            --Maximum 4 SMUs are supported; if not 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 not 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 not input 'nil'
```



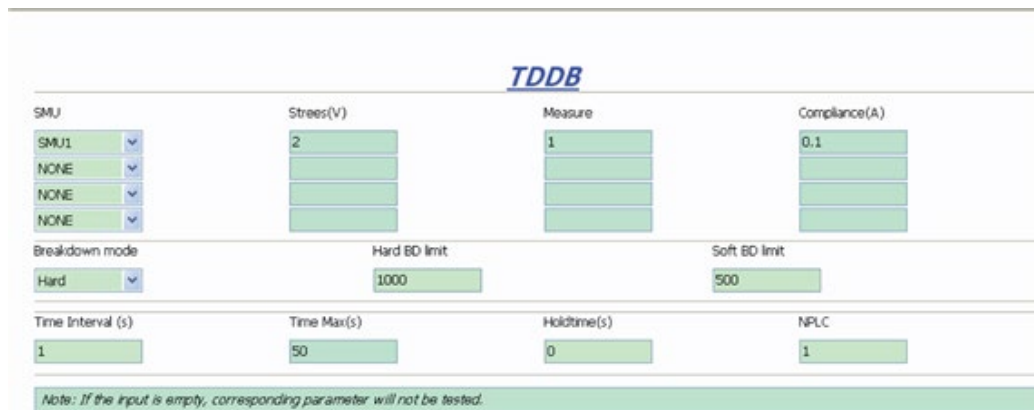
```

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 not 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
    
```

**GUI-related**

The next Figure shows the dialog box for the TDDB test. A general description of this dialog box is included below.

**Figure 63: GUI for TDDB**



Terminal setting: If the SMU is set to NONE in the SMU list, you will 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 settings: If Breakdown mode is set to Hard, the Soft Breakdown can be left empty. For Soft Breakdown details, see next Figure JESD92.

Time arrangement: Time Max can be left empty. In this case, the test will go on until all devices fail.

**Example call:**

```
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)
```

## NBTI

The negative bias temperature instability (NBTI) script is used to perform the NBTI test. It 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/drain-current).

### Syntax:

```
NBTI(smu_D, smu_G, smu_S, smu_B, vg_stress, vd_stress, vg_meas, vd_meas,
myNPLC, meas_delay, navg, t_array, modeflag, complianceni, 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 ~ 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 complianceni=0.1 in [0,] -- Current compliance
```

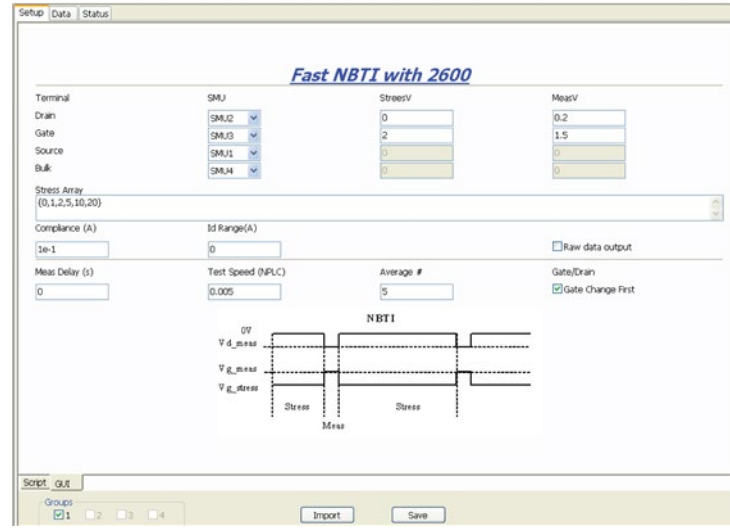
### Outputs:

```
time={} -- Time table
did={}  -- Drain-current shift table
```

### GUI-related

The next Figure shows the NBTI dialog box and illustrates the testing method. A general description of this dialog box is included below.

Figure 64: GUI for NBTI



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 entry field sets the time before each measurement. The Test Speed entry field sets the PLC value. The Average # entry field decides the number of measurements taken.

Gate/Drain: Voltages are applied on the gate and drain and change when the measurement begins and ends. The gate/drain selection box is used to determine which terminal will change first. If Gate Change First is selected, the gate terminal changes first. If the Gate Change First is deselected, the drain terminal changes first.

Stress Array: Used to input the time array.

#### Example call:

```

local complianci=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,complianci,time,did)

```

## NBTI\_meas

This module performs the negative bias temperature instability (NBTI) test, with pre-Id\_Vg testing and post-Id\_Vg testing.

### Syntax:

```
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, Gcomp, Dcomp, 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 of idvg test. "1" meas enable pre/post idvg test, "0" meas disable it
integer flag1=1 in [0,1]	-- flag of NBTI test. "1" means enable NBTI stress-measure test, "0" means disable it
integer flag2=1 in [0,1]	-- flag of Vcti test. "1" meas enable Vcti test, "0" meas disable it
double p_Vg_lo=0 in [-40, 40]	-- start of gate-voltage sweep in pre/post test
double p_Vg_hi=2 in [-40, 40]	-- stop of gate-voltage sweep in pre/post test
double p_Vg_points=21 in [0, 4096]	-- gate-voltage sweep number of points in pre/post test
double p_Vds=1 in [-40, 40]	-- drain-source bias in pre/post test
double p_Drangei=1e-3 in [0, 0.1]	-- drain-current range in pre/post test
double p_sweepdelay=0 in [0,]	-- sweep delay in pre/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,]	-- wide 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 ~ 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 trian pulses
integer t_mode=1 in [0,2]          -- "0" for time array given by customer; "1" for
                                   logarithmic time; "2" for 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,]        -- means number-of-point-per-decade when
                                   t_mode is 1; means delta time when t_mode is 2
table time_input in [0,]          -- if t_mode is 0, this array will be taken as 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 sampling.

```

**Outputs:**

```

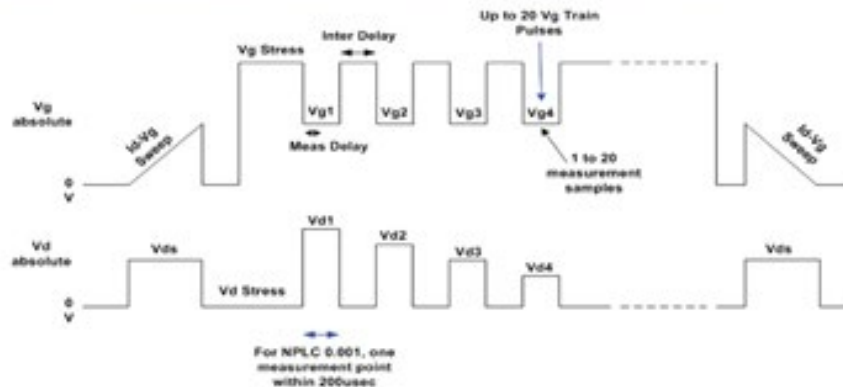
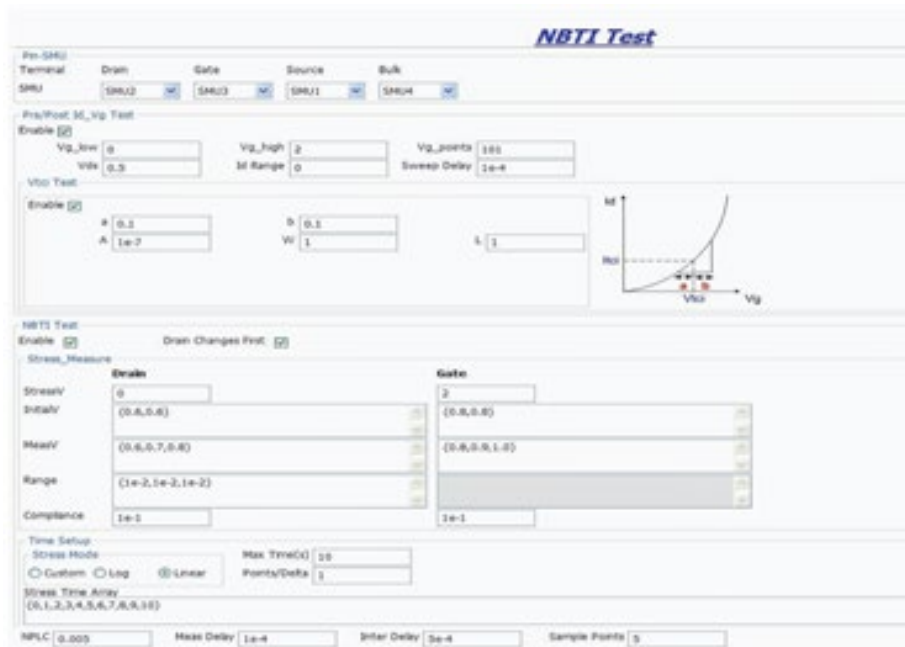
error      -- working condition flag
Vg_pre     -- gate-voltage of pre test
Id_pre     -- drain-current of pre test
Vg_pos     -- gate-voltage of post test
Id_pos     -- drain-current of post test
Vtci      -- gate-voltage at target drain-current
Idini     -- initial current of drain
Idend     -- drain-current after stress sequence
time      -- time table
Id1       -- drain-current table
Id2
Id3
Id4
Id5
Id6
Id7
Id8
Id9

```

- Id10
- Id11
- Id12
- Id13
- Id14
- Id15
- Id16
- Id17
- Id18
- Id19
- Id20

The next Figure shows the NBTI\_meas test dialog box and illustrates the testing method. A general description of this dialog box is included below.

Figure 65: GUI for NBTI\_meas



**Example call:**

```
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_Vd
s,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,G
compi,Dcompi,rng,Nsam)
```



## NBTI\_on\_the\_fly

- Reference: "on-the-fly characterization of NBTI in ultra-thin gate oxide PMOSFETs," M. Denais, et. al, IEDM 2004.
- The code is a Keithley Instruments, Inc. copyright.
- This is a new methodology for monitoring threshold voltage degradation and relaxation for NBTI and charge trapping on high K gate stacks.
- Vg\_stress is for stress and measurement during the stress phase.
- Vg\_relax is for measurement during recovery.
- 0 is set for recovery voltage during time other than measurement.
- This test can only be used for one device: once during the "stress-on" period and once during the "stress-off" period.

### Possible outputs:

```
'ERROR' (possible error type)           --1 stands for 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
```

### Syntax:

```
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: monitoring 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 as 0

double Vg_dist=0.05         -- Delta Vg for different Id measurement

double Vd=0.1                -- Drain-voltage only applied during monitoring, other times = 0V

integer Stress_time=1000     -- Time for stress in seconds

table Monitor_time_stamp={ } -- Time in seconds; this is an input array for guiding time
between two monitorings; the actual time stamp for monitoring
might not be exactly the same due to measurement time; also,
this time stamp is the same for both stress on (degradation
monitoring) and off (relaxation monitoring)

instid GSMU=SMU1             -- Gate SMU number, SMU1 for example

instid DSMU=SMU2             -- Drain SMU number, SMU2

instid SSMU=KI_GND           -- source SMU number

instid BSMU=KI_GND           -- bulk SMU number
```

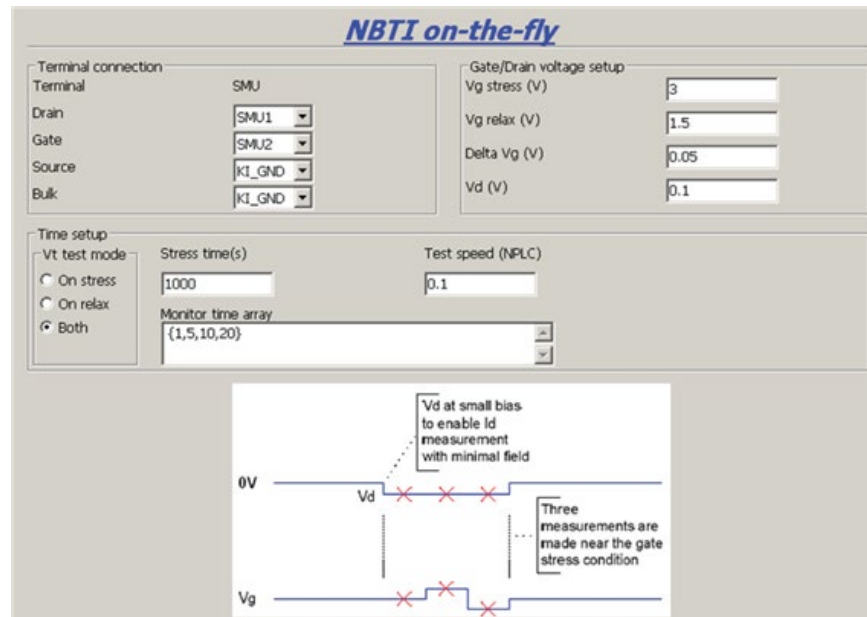
```
double myNPLC=0.01
```

```
-- PLC setting
```

### GUI-related

The next 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 66: GUI for NBTI\_on\_the\_fly**



Terminal connection: SMUs are assigned to terminals source and bulk and KI\_GND is manually set. If specific SMUs are assigned to these two terminals, 0V will be 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: Arranges time during the stress and relaxation. 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 decided 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 call:

```
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)
```

## qbd\_rmpj

Function: Performs a charge-to-breakdown test using the QBD Ramp J test algorithm described in JESD35-A, "Procedure for Wafer-level Testing of Thin Dielectrics." This algorithm forces a logarithmic current ramp until the oxide layer breaks down. This algorithm is capable of a maximum current of +/- 1A, if a high power SMU is used.

### Syntax:

```
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 not input '0'
integer LoSMUId1=0 in[0,1,2..64]         --maximum 4 smus are supported. if not input '0'
integer LoSMUId2=0 in[0,1,2..64]         --maximum 4 smus are supported. if not input '0'
integer LoSMUId3=0 in[0,1,2..64]         --maximum 4 smus are supported. if not 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 10uA/cm^2 and may change
                                         depending oxide area; For maximum sensitivity the
                                         specified value should be well above the worse 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 s

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 will exit. 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). Used to terminate a test where
                                         breakdown occurs but was not detected during
                                         the test.
```

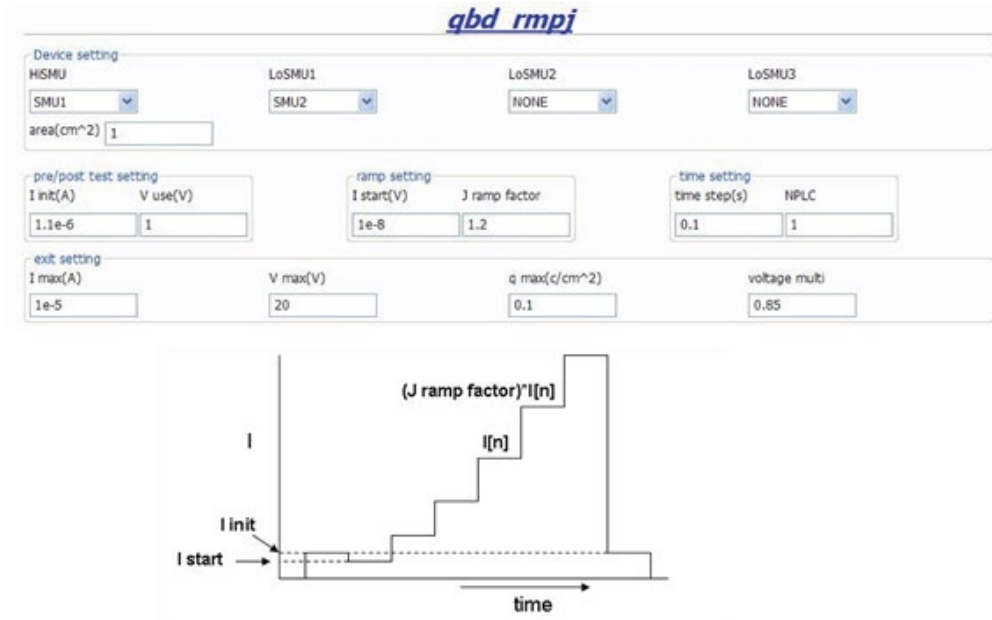
```
double area=2 in(0,) --area of oxide structure (cm^2)
```

**Outputs:**

```
V_stress      --voltage array
I_stress      --current array
T_stress      --time stamp array representing when current is measured
q_stress      --accumulated charge array PER OXIDE AREA
V_init_pre    --voltage at I_init in pre test
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          --time stamp when measuring I_bd
Failure_mode  --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: non-Catastrophic (initial test pass, ramp test fail, post test pass)
              --5: Others (initial test pass, ramp test pass, post test pass)
Test_status   --0: no test errors (exit due to measured voltage < exit_volt_mult*V_previous)
              --(-1): failed pre-stress test
              --(-2): cum charge limit reached
              --(-3): current limit reached
              --(-4): voltage limit reached
              --(-5): masked Catastrophic Failure
              --(-6): non-Catastrophic Failure
              --(-7): Invalid specified t_step
```

The next Figure shows the QBD Ramp J dialog box and illustrates the testing method.

Figure 67: GUI for qbd-rmpj



**NOTE**

If the above routine is modified, change the function name to avoid possible programming errors.

**Example call:**

```

local HiSMUIId=1
local LoSMUIId1=2
local LoSMUIId2=0
local LoSMUIId3=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(HiSMUIId, LoSMUIId1, LoSMUIId2, LoSMUIId3, myplc, v_use, I_init,
I_start, F, t_step, exit_volt_mult, V_max, I_max, q_max, area).
    
```

## qbd\_rmpv

**Function:** Performs a charge-to-breakdown test using the QBD Ramp V Test algorithm described in JESD35-A, "Procedure for Wafer-level Testing of Thin Dielectrics." This algorithm forces a linear voltage ramp until the oxide layer breaks down. This algorithm is capable of a maximum voltage of +- 200 volts.

### Syntax:

```
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 not input '0'
integer LoSMUId1=0 in[0,1,2..64] --maximum 4 smus are supported. if not input '0'
integer LoSMUId2=0 in[0,1,2..64] --maximum 4 smus are supported. if not input '0'
integer LoSMUId3=0 in[0,1,2..64] --maximum 4 smus are supported. if not 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.
Typical value is 10uA/cm^2 and may change depending
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 Vuse 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 0.1MV/cm, for example, the maximum value can
be calculated using Tox*0.1MV/cm, where Tox is in unit
of centimeters. This is 0.1V for a 10nm oxide.
double t_step=0.1 in(0,) --Voltage ramp step time(Unit:s). This is used to
determine the voltage ramp rate; This time should be
less or equal than 100ms. Typically 40 - 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. A typical value is 1uA.
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 prior to breakdown.
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, will result in failure; recommended value: 10-100.
double exit_slope_mult=3 in(0,)	--Change of slope failure criteria. This is the factor of change in FN slope, which, if exceeded, will result in failure; recommended value: 3.
double q_max=100 in(0,)	--Maximum accumulated oxide charge PER OXIDE AREA! Used to terminate a test where breakdown occurs but was not detected during the test. (C/cm <sup>2</sup> ).
double t_max=10 in(0,)	--maximum stress time allowed(Unit:s); Reaching the limit will result in test finish.
double v_max=2 in(-200,200)	--The maximum voltage limit for the voltage ramp. This limit is specified at 30MV/cm for oxides less than 20nm thick and 15MV/cm for thicker oxides. For example, v_max can be estimated from Tox*30Mv/cm where Tox is in centimeters. This is 35V for a 10.0nm Oxide.
double area=2 in(0,)	--area of oxide structure (cm <sup>2</sup> )
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).

**Outputs:**

V_stress	--voltage stress array
I_stress	--measured current array
T_stress	--time stamp array representing when current is measured
q_stress	--accumulated charge array PER OXIDE AREA
I_use_pre	--Measured oxide current at v_use prior to starting the ramp
I_use_post	--Measured oxide current at v_use after the ramp finished
Q_bd	--Charge to breakdown. Cumulative charge passing through the oxide prior to breakdown (C)
q_bd	--charge to breakdown density (C/cm <sup>2</sup> )
v_bd	--applied voltage at the step just before oxide breakdown
I_bd	--measured current at v_bd just before oxide breakdown
t_bd	--time stamp 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

Failure\_mode --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. non-Catastrophic (initial test pass, ramp test fail, post test pass)

--5. Others (initial test pass, ramp test pass, post test pass)

Test\_status --2: no test errors (exit due to measured current > exit\_curr\_multi\*I\_previous)

--1: no test errors (exit due to measured current > calculated failure slope ONLY)

--0: no test errors (exit due to measured current > I\_max ONLY)

--(-1): failed pre-stress test

--(-2): cumulative charge limit reached

--(-3): voltage limit reached

--(-4): maximum time limit reached

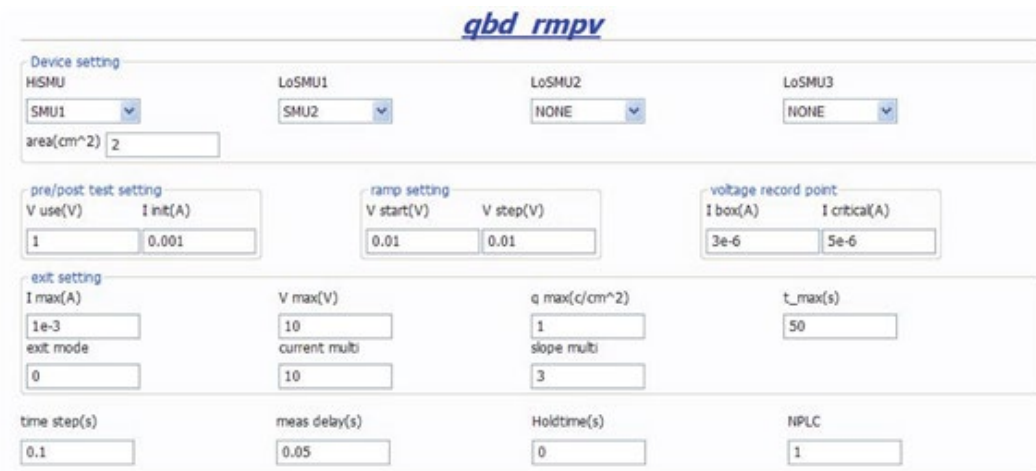
--(-5): masked Catastrophic Failure

--(-6): non-Catastrophic Failure

--(-7): Invalid specified t\_step, hold\_time or measure\_delay

The next Figure shows the QBD Ramp V dialog box and illustrates the testing method.

**Figure 68: GUI for qbd\_rmpv**





**Example call:**

```
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).
```

## Em\_iso\_test

Isothermal EM description: This script is used to run JEDEC 61-compliant isothermal Electromigration tests. Data will be periodically printed.

### Syntax:

```
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           --No. of SMUs (choose 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         --(*C) reference temperature for TCR
double Vlimit = 40       --Voltage limit (V)
double Ilimit = 1        --Current limit (A)
double init_failR       --Room temperature failure resistance (Ohm)
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 & before measure (typical: 50-100ms)
double equil_time = 3    --(s) maximum time for convergence control loop
double Ttarget = 300     --(*C) target temp of metal line
double Terror = 0.5      --Error band allowed to control temperature (*C)
double Rfail_fact = 1.5  -- % 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 DUT 1, s
source_smu1    --Source value for DUT 1, A
```

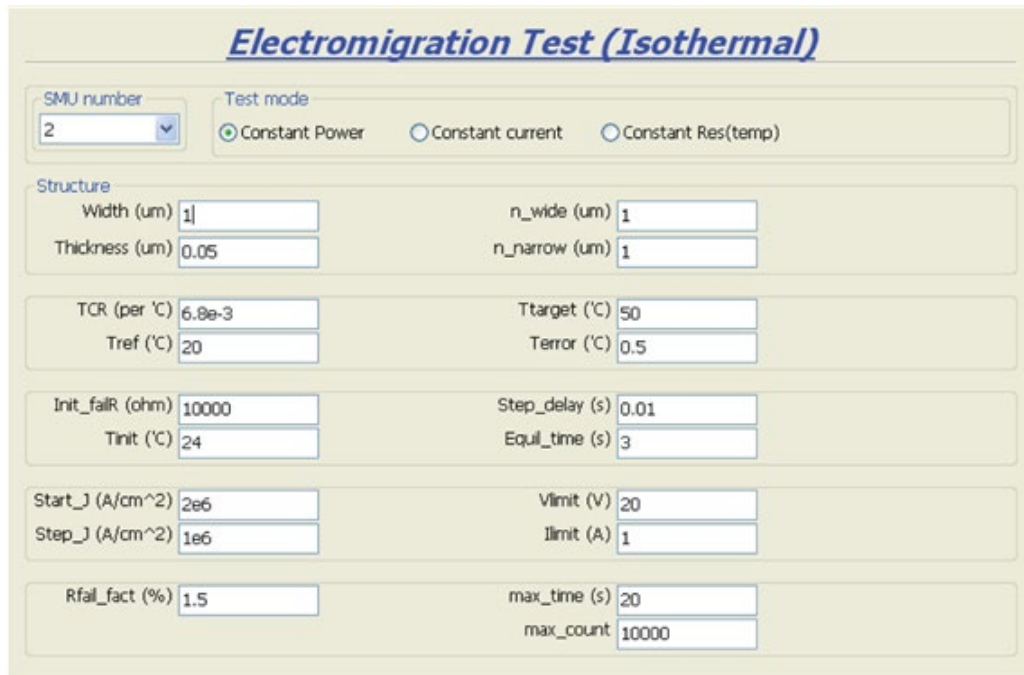
```

Reading_smu1  --Readings for DUT 1, ohm
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, ohm
Temp_smu2    --Readings for DUT 2, ohm
Time_smu3    --Timestamp for DUT 3, s
source_smu3  --Source value for DUT 3, A
Reading_smu3 --Readings for DUT3, ohm
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, ohm
Temp_smu4    --Temperature for DUT 4, K
    
```

**GUI-related:**

The next Figure shows the GUI for the Iso\_EM test.

**Figure 69: GUI for ISO\_EM test**



**Example Call:**

```
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)
```

## Python user library introduction

ACS has a Python user library (PTMLib), which includes CV test, matrix control, scope control and other external instrument libraries. It is located 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 details about how to import a PTM user module, refer to the Configuring a Python Language Test Module (PTM) topic in the ACS Reference Manual (document number: ACS-901-01).

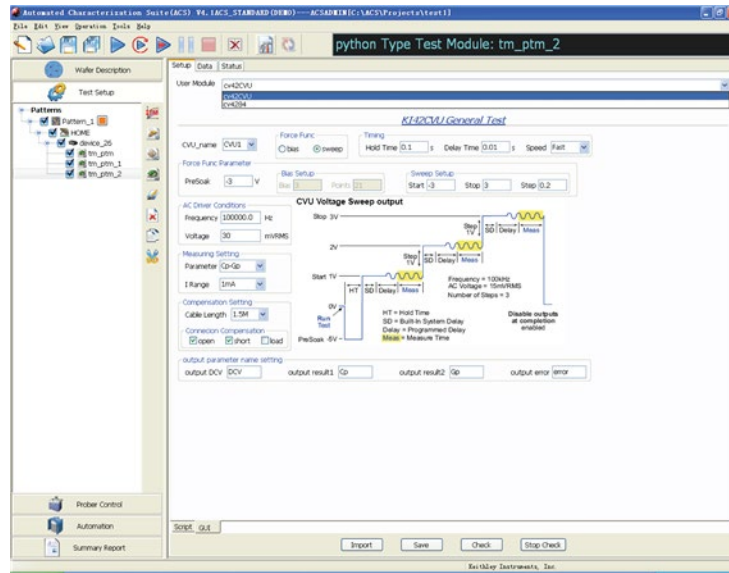
## Configure a capacitor meter library

### CV\_4200CVU

These modules are used to test capacitive parameters at specified frequencies and AC drive voltages, with measurements at the DC voltage bias or sweep using the Model 4200-CVU.

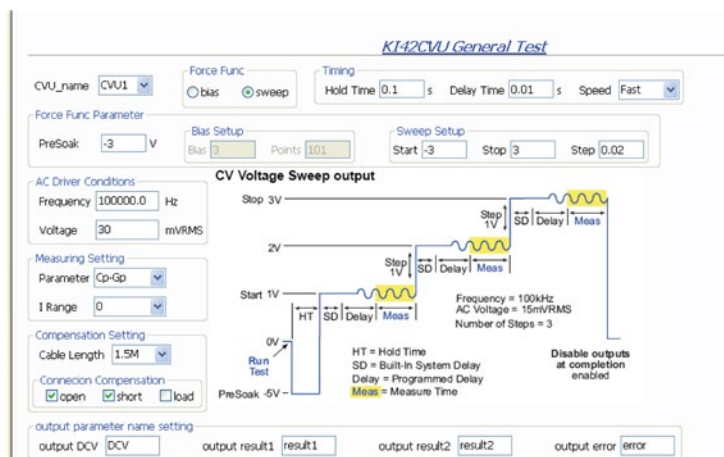
Add a PTM to the configuration navigator, then import the CVITM.py module from the PTMLib library. The CV test GUI will display. Click the CVU4200 in the user module to open the Model 4200-CVU test (see next Figure).

Figure 70: Select 4200CVU user module



The details of the Model 4200-CVU GUI are indicated in the graphic with a description that follows (see next Figure).

Figure 71: CV\_4200CVU setting example



The GUI inputs are as follows:

CVU\_name: Instrument ID of the Model 4200-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 to 999s)
- 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 clicking bias in Force Func)

- Bias: Force value for the bias
- Points: The number of bias points

Sweep Setup (enabled by clicking 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: 10kHz to 100kHz in 10kHz steps, 100kHz to 1MHz in 100kHz steps, 1MHz to 10MHz in 1MHz steps. If an entered value is not a supported frequency, the closest supported frequency will be selected (for example, 15kHz input will change to 20kHz).
- Voltage: Voltage level of the AC drive (10mV to 100mVRMS).

**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 auto range.

**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 desired name for the output parameter

**Output error list:**

- 0: OK
- 10000: Specified CVU does not exist
- 10001: (INVAL\_PARAM) Parameter setting error occurred
- 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

This module is used to test capacitive parameters at specified frequencies and AC drive voltages, with measurements at the DC voltage bias or sweep.

Add a PTM to the configuration navigator, then import the CVITM.py module from the PTMLib library. The CV test GUI will display. Click the CV4284 in the user module to open the Model 4200-CVU test (see next two Figures).

Figure 72: Select HP4284 user module

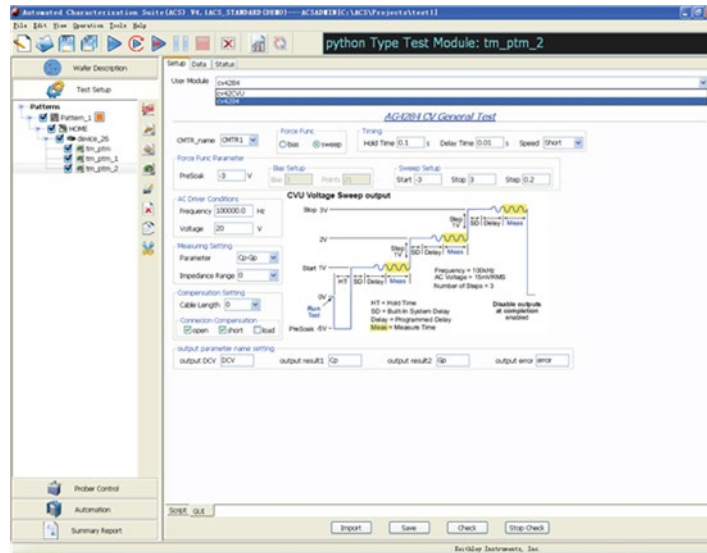
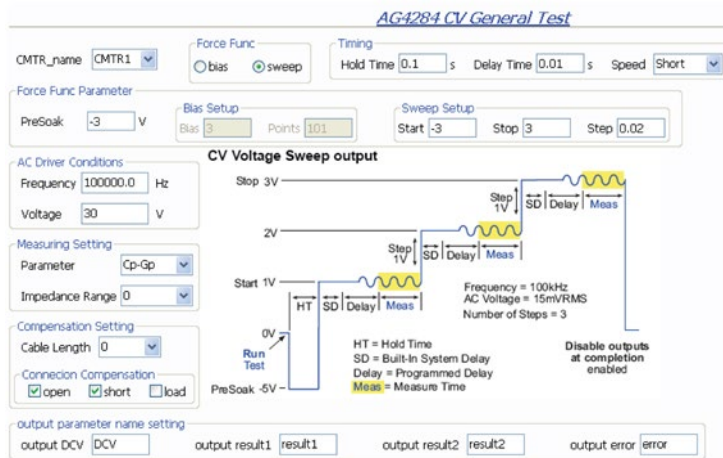


Figure 73: CV\_HP4284 setting example





The GUI inputs are as follows:

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 clicking bias in Force Func)

- Bias: Force value for the bias
- Points: The number of bias points

Sweep Setup (enabled by clicking 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: 10kHz to 100kHz in 10kHz steps, 100kHz to 1MHz in 100kHz steps, 1MHz to 10MHz in 1MHz steps. If an entered value is not a supported frequency, the closest supported frequency will be selected (for example, 15kHz input will change to 20kHz).
- Voltage: Voltage level of the AC drive (10mV to 100mVRMS).

**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 desired name for the output parameter

**Output error list:**

- 0: OK
- 10000: Specified CVU does not exist
- 10001: (INVAL\_PARAM) Parameter setting error occurred
- 10090: (GPIB\_ERROR\_OCCURRED) A GPIB communications error occurred

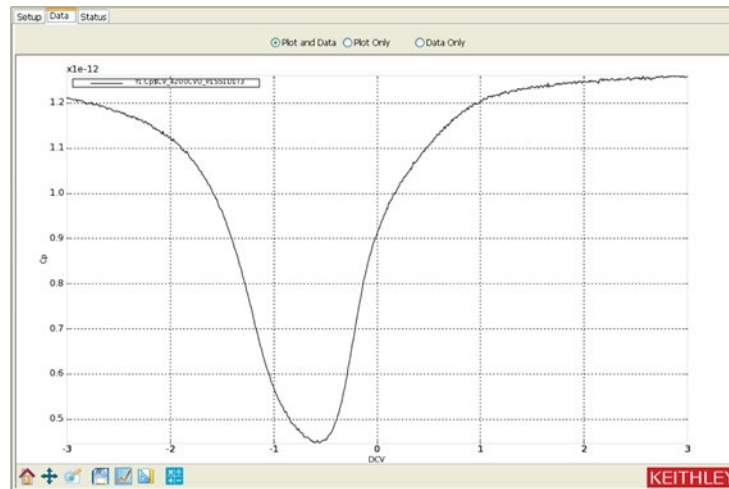
*return dictionary:*

- 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_t
ime, delay_time, speed, freq_bias, v_AC, meas_param, meas_range, cable_length, isCm
pstOpen, isCmpstShort, isCmpstLoad, output_DCV, output_result1, output_result2, o
utput_error)
```

**Figure 74: CV\_HP4284 Data tab**

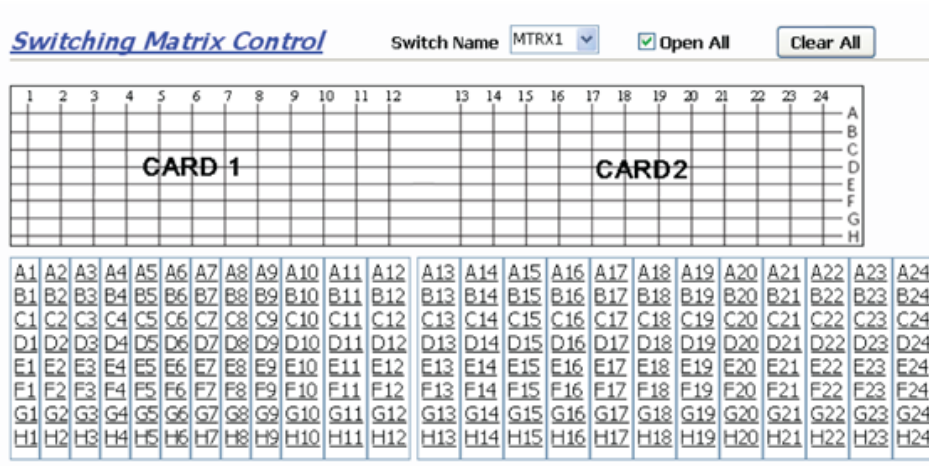


## Configure a switch matrix library

### Switch\_Control

Add a PTM to the configuration navigator, then import the swichctrl.py module from the PTMLib library. The Switch GUI will display (see next Figure).

Figure 75: Switch\_Control GUI



This module connects matrix row terminals and column pins, according to the row list and column list. It supports two cards in one switch controller at maximum.

Instrument: Keithley Instruments Switch Matrix 707A, 708A

NOTE

The Model 708A module can control two cards at most. Plus, the card involved does not need to be configured in the hardware configuration panel.

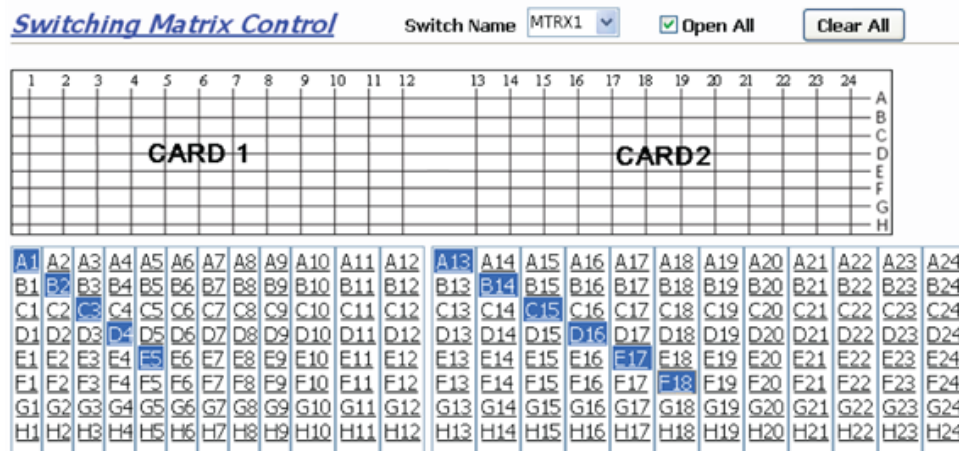
**Inputs:**

- 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 which will be closed.['A','B']
- collist (list): Matrix column name which will be closed.['1','2']

In the GUI, you can control the matrix:

- Select the switch matrix in the drop-down list for the Switch Name
- Click the cells on the panel and the related rows and columns of the matrix will connect. For example, click A1, and the 1 column and A row will connect. The corresponding cell will highlight (see next Figure). Click the highlighted cells again, and the connections will be cancelled.
- 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 76: Switch\_Control setting example**



## KISeries 3700 System Switch

Add a PTM to the configuration navigator, then import the MDD.py module from the PTMLib library. The 3700 Matrix GUI will display (see next two Figures).

Instrument: Keithley Instruments Series 3700 System Switch/Multimeter Cards

This module supports two types of cards: 6x16, High Density, Matrix Card (Model 3730) and Dual 1x30 Multiplexer Card (3720).

Figure 77: Series 3700 System GUI

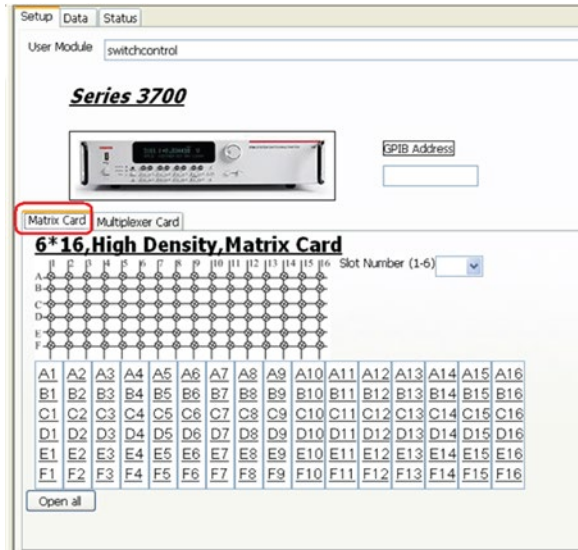
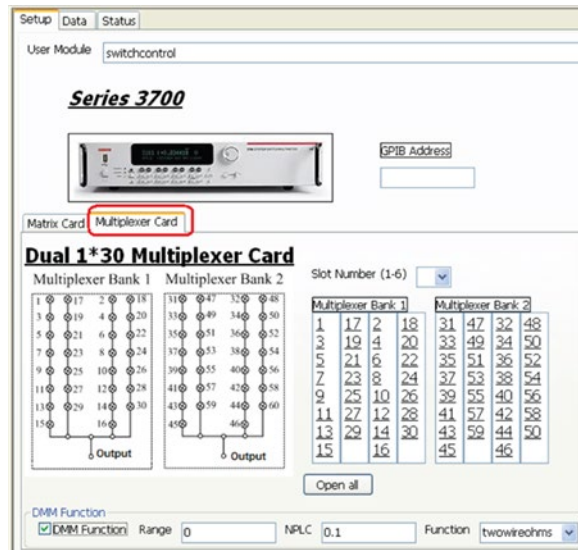


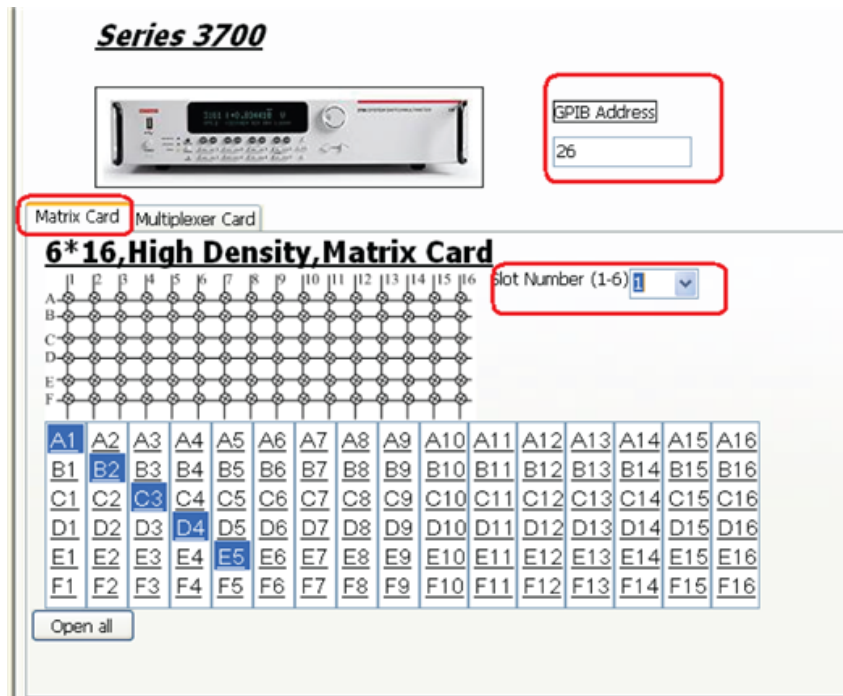
Figure 78: Series 3700 System Multiplexer GUI



To control the Model 3700 matrix from the GUI (see next Figure):

1. Input the GPIB address number in the GPIB edit box.
2. Click the Matrix Card tab.
3. Select the slot number from 1 to 6.
4. Click the cells on the panel and the related rows and columns of the matrix will connect. For example, click A1, and the 1 column and the A row will connect. The corresponding cell will highlight (see next Figure). Click the highlighted cells again, and the connections will be cancelled.
5. If you want to clear all connections, click the Open all button.

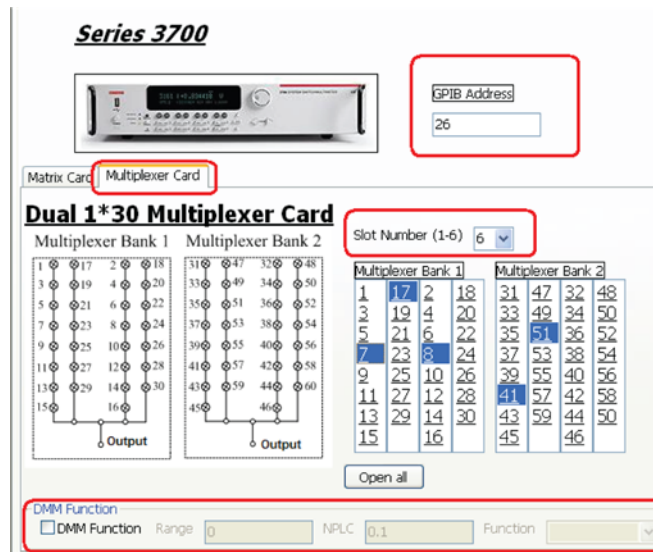
**Figure 79: Matrix control setting example**



To control the multiplexer card from the GUI (see next Figure):

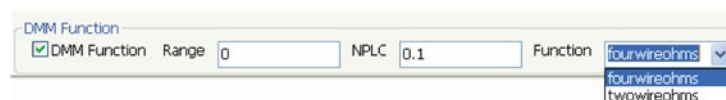
1. Input the GPIB address number in the GPIB edit box.
2. Click the Multiplexer Card tab.
3. Select the slot number from 1 to 6.
4. Click the cells on the panel and the related rows and columns of the matrix will connect. For example, click the A1, and the 1 column and the A row will connect. The corresponding cell will highlight (see next Figure). Click the highlighted cells again, and the connections will be cancelled.
5. If you want to clear all connections, click the Open all button.

**Figure 80: Multiplexer control setting example**



If want to accomplish a DMM function test, select the DMM Function, then set the range (0 means auto-range), NPLC, and select the function (see next Figure).

**Figure 81: 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

### TEKSCOPE\_ReadWave

Add a PTM to the configuration navigator, then import the TEKSCOPE.py module from the PTMLib library. The TEK SCOPE GUI will display (see next two Figures).

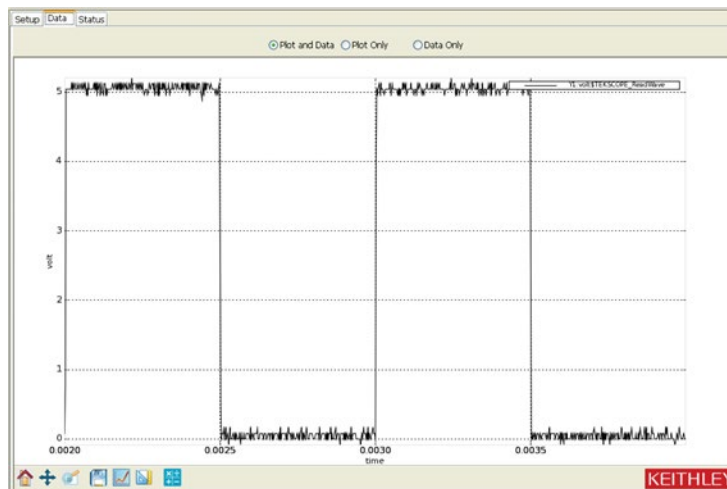
This module reads data from one channel at a time. Some modifications are needed in order to enable it to read data from more channels simultaneously.

Instrument: TEKSCOPE

Figure 82: TEKSCOPE read wave test module GUI



Figure 83: Waveform reading data



## Configure a Series 23x library

### BiasVolt\_SampleCurr

This module is used to bias voltage and take current readings for Models 236/237/238.

Instrument: Keithley Instruments Model 236/237/238 source measure unit.

#### Inputs:

instAddr:	GPIO 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 auto-range.
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 auto-range.

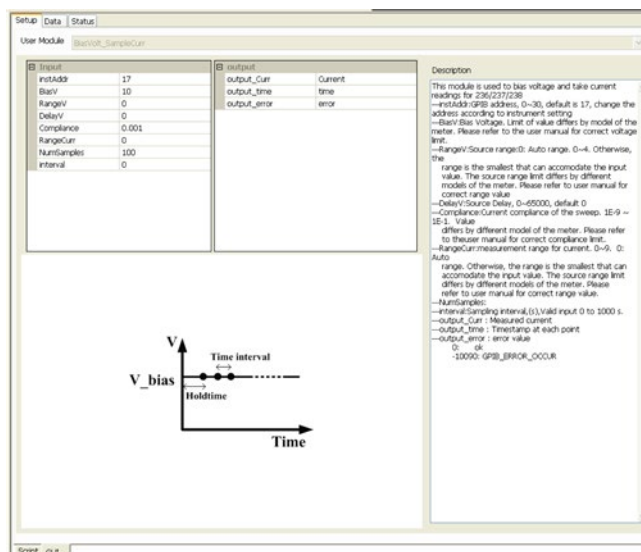
#### NumSamples:

interval: Sampling interval. Valid input is zero to 1000 seconds.

#### Outputs:

output_Curr:	Measured Current output.
output_time:	Timestamp at each point.
output_error:	Error value
0:	OK
-10090:	GPIO_ERROR_OCCUR

Figure 84: 23x Bias V Sample standard GUI



## Sweepssystem computer\_23x

---

This module is used to sweep current and take current, voltage, and time readings for the Model 236/237/238.

Instrument: Keithley Instruments Model 236/237/238 source measure units.

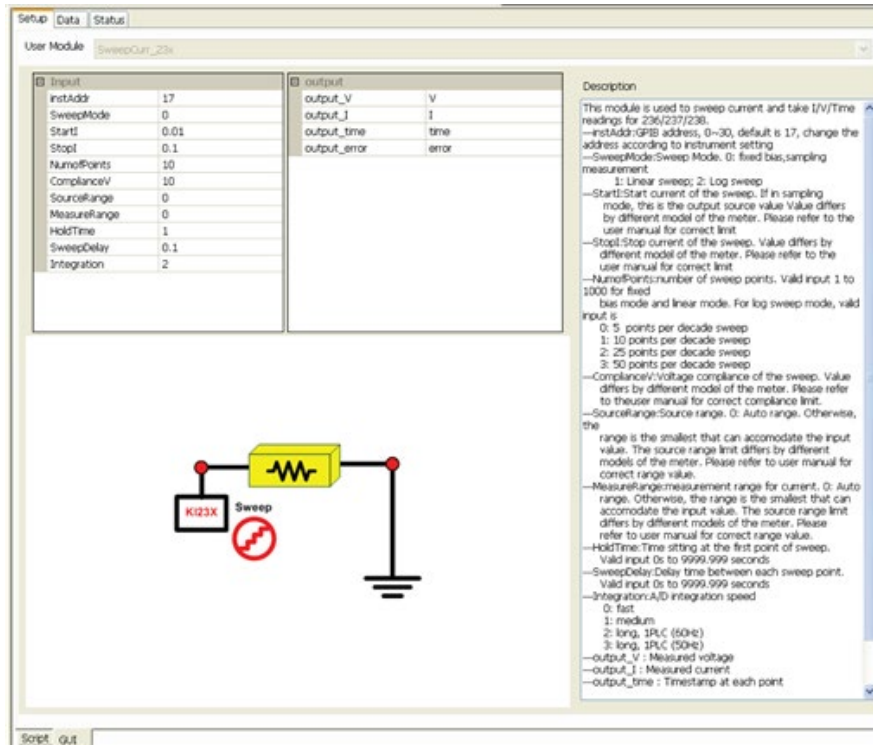
### Inputs:

<code>instAddr:</code>	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting.								
<code>SweepMode:</code>	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep.								
<code>StartI:</code>	Start the sweep current. If in sampling mode, this is the output source value.								
<code>StopI:</code>	Stop the sweep current.								
<code>NumofPoints:</code>	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: <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>5 points per decade sweep</td> </tr> <tr> <td>1:</td> <td>10 points per decade sweep</td> </tr> <tr> <td>2:</td> <td>25 points per decade sweep</td> </tr> <tr> <td>3:</td> <td>50 points per decade sweep</td> </tr> </table>	0:	5 points per decade sweep	1:	10 points per decade sweep	2:	25 points per decade sweep	3:	50 points per decade sweep
0:	5 points per decade sweep								
1:	10 points per decade sweep								
2:	25 points per decade sweep								
3:	50 points per decade sweep								
<code>ComplianceV:</code>	Voltage-sweep compliance.								
<code>sourceRange:</code>	Source range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.								
<code>MeasureRange:</code>	Measurement range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.								
<code>HoldTime:</code>	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 seconds.								
<code>SweepDelay:</code>	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.								
<code>Integration:</code>	Analog/digital integration speed: <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>fast</td> </tr> <tr> <td>1:</td> <td>medium</td> </tr> <tr> <td>2:</td> <td>long, 1PLC (60Hz)</td> </tr> <tr> <td>3:</td> <td>long, 1PLC (50Hz)</td> </tr> </table>	0:	fast	1:	medium	2:	long, 1PLC (60Hz)	3:	long, 1PLC (50Hz)
0:	fast								
1:	medium								
2:	long, 1PLC (60Hz)								
3:	long, 1PLC (50Hz)								

### Outputs:

<code>output_V:</code>	Measured voltage						
<code>output_I:</code>	Measured current						
<code>output_time:</code>	Timestamp at each point						
<code>output_error:</code>	Error value <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>OK</td> </tr> <tr> <td>-10090:</td> <td>GPIB_ERROR_OCCUR</td> </tr> <tr> <td>-10100:</td> <td>INVAL_PARAM</td> </tr> </table>	0:	OK	-10090:	GPIB_ERROR_OCCUR	-10100:	INVAL_PARAM
0:	OK						
-10090:	GPIB_ERROR_OCCUR						
-10100:	INVAL_PARAM						

Figure 85: 23x Sweep standard GUI



## SweepVolt\_23x

---

This module is used to sweep voltage and take current, voltage, and time readings for the Model 236/237/238.

Instrument: Keithley Instruments Model 236/237/238 source measure units.

### Inputs:

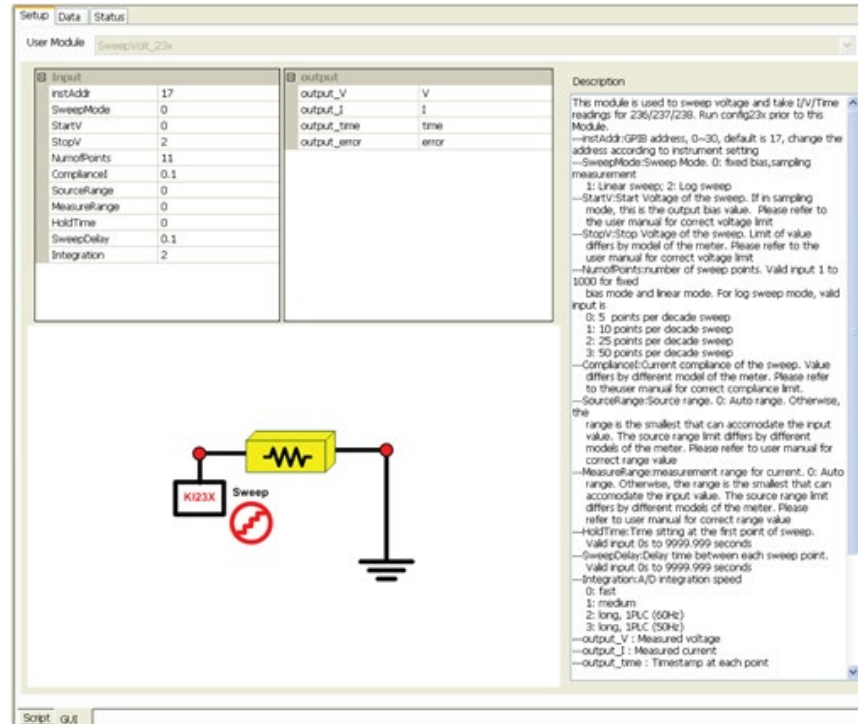
<code>instAddr:</code>	GPIB address, 0 through 30; default is 17; change the address according to the instrument setting.								
<code>SweepMode:</code>	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep.								
<code>StartV:</code>	Start the sweep voltage. If in sampling mode, this is the output bias value.								
<code>StopV:</code>	Stop the sweep voltage.								
<code>NumofPoints:</code>	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: <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>5 points per decade sweep</td> </tr> <tr> <td>1:</td> <td>10 points per decade sweep</td> </tr> <tr> <td>2:</td> <td>25 points per decade sweep</td> </tr> <tr> <td>3:</td> <td>50 points per decade sweep</td> </tr> </table>	0:	5 points per decade sweep	1:	10 points per decade sweep	2:	25 points per decade sweep	3:	50 points per decade sweep
0:	5 points per decade sweep								
1:	10 points per decade sweep								
2:	25 points per decade sweep								
3:	50 points per decade sweep								
<code>ComplianceI:</code>	Current-sweep compliance.								
<code>sourceRange:</code>	Source range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.								
<code>MeasureRange:</code>	Measurement range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.								
<code>HoldTime:</code>	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 seconds.								
<code>SweepDelay:</code>	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.								
<code>Integration:</code>	Analog/digital integration speed: <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>fast</td> </tr> <tr> <td>1:</td> <td>medium</td> </tr> <tr> <td>2:</td> <td>long, 1PLC (60Hz)</td> </tr> <tr> <td>3:</td> <td>long, 1PLC (50Hz)</td> </tr> </table>	0:	fast	1:	medium	2:	long, 1PLC (60Hz)	3:	long, 1PLC (50Hz)
0:	fast								
1:	medium								
2:	long, 1PLC (60Hz)								
3:	long, 1PLC (50Hz)								

### Outputs:

<code>output_V:</code>	Measured voltage						
<code>output_I:</code>	Measured current						
<code>output_time:</code>	Timestamp at each point						
<code>output_error:</code>	Error value <table style="margin-left: 40px;"> <tr> <td>0</td> <td>OK.</td> </tr> <tr> <td>-1</td> <td>23x not found on GPIB</td> </tr> <tr> <td>-10000</td> <td>(INVAL_INST_ID) The specified instrument ID does not exist.</td> </tr> </table>	0	OK.	-1	23x not found on GPIB	-10000	(INVAL_INST_ID) The specified instrument ID does not exist.
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 86: 23x SweepV standard GUI



## Vdsls\_237

---

High-voltage measurement of the drain current while forcing drain voltage and stepping the gate voltage.

Instruments: Keithley Instruments Model 236/237/238 source measure unit and Model 4200 Semiconductor Characterization System.

### Inputs:

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: <table style="margin-left: 40px;"> <tr> <td>0:</td> <td>fast</td> </tr> <tr> <td>1:</td> <td>medium</td> </tr> <tr> <td>2:</td> <td>long, 1PLC (60Hz)</td> </tr> <tr> <td>3:</td> <td>long, 1PLC (50Hz)</td> </tr> </table>	0:	fast	1:	medium	2:	long, 1PLC (60Hz)	3:	long, 1PLC (50Hz)
0:	fast								
1:	medium								
2:	long, 1PLC (60Hz)								
3:	long, 1PLC (50Hz)								
DelayTime:	Delay time of measurements (in 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.								
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.

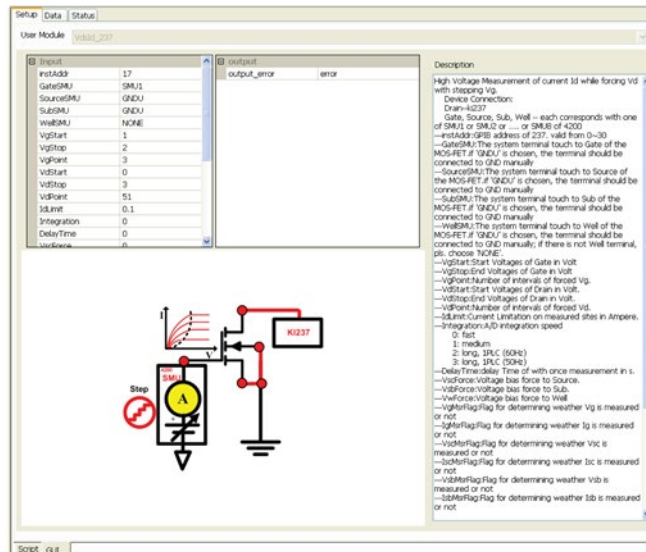
IwMsrFlag: Flag to determine if the well current is measured.

**Outputs:**

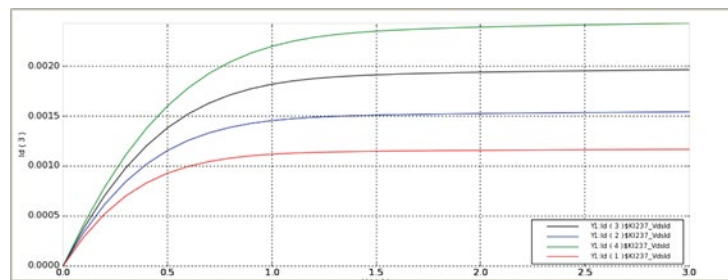
output\_error: Error value

- 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 87: Model 237 VdsId standard GUI**



**Figure 88: Model 237 VdsId test result**



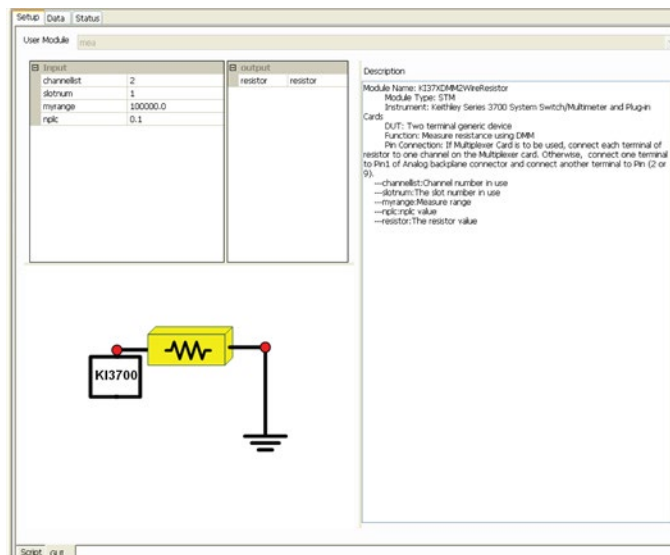


## Configure a Series 3700 system switch DMM library

### Series 3700 System Switch DMM two-wire

Add a PTM to the configuration navigator, then import the Gpibresistor.py module from the PTMLib library. The 3700 DMM resistor test GUI will display (see next Figure).

**Figure 89: 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.

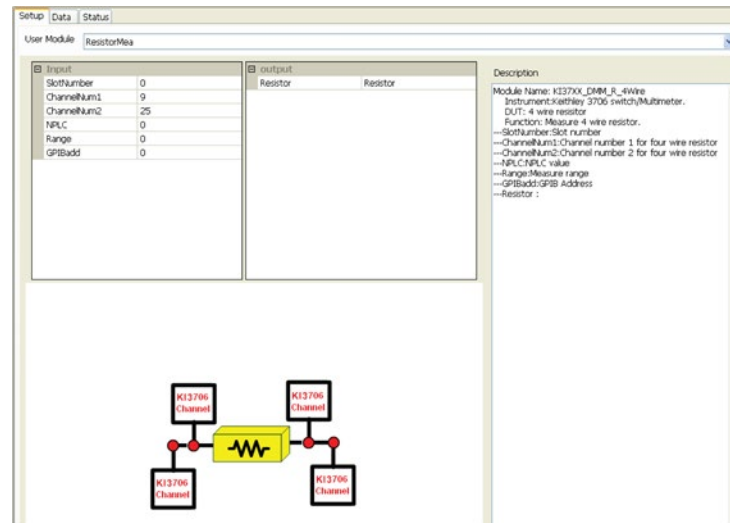
Function: Measure resistance using a DMM.

Pin Connection: 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

Add a PTM to the configuration navigator, then import the FourWireResistor.py module from the PTMLib library. The 3700 DMM resistor test GUI will display (see next Figure).

**Figure 90: 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 Connection: 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

Add a PTM to the configuration navigator, then import the HiPower\_24.py module from the PTMLib library. The Series 2400 SourceMeter test library GUI will display. Choose the appropriate test module from Test Module drop-down list.

## Series 2400 drain-current test

---

Instrument: Keithley Instruments Model 2430 SourceMeter.

Function: This module is used to test the drain-current at a specified drain-voltage, during a gate-voltage sweep.

Pin Connection: 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.
- Get the  $V_{tx}$  and  $V_{t0}$  results.

### Input:

<code>drain_addr</code> (int):	Model 2430 GPIB drain-terminal address.
<code>gate_addr</code> (int):	Series 2400 SourceMeter GPIB gate-terminal address.
<code>vg_start</code> (double):	Start the gate-voltage pulse (valid input is from -200V to 200V).
<code>vg_stop</code> (double):	Stop the gate-voltage pulse (valid input is from -200V to 200V).
<code>points</code> (int):	Number of drain-sweep points (valid input 1 to 2500).
<code>vd</code> (double):	Drain the bias voltage.
<code>hold_time</code> (double):	Hold time in second before gate sweep. Valid inputs are zero to 9999.999 seconds.
<code>delay_time</code> (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.
<code>limiti</code> (double):	Drain-voltage force compliance value (valid input is from -10A to 10A).
<code>rangei</code> (double):	Range for drain-current measurement. For pulse mode, auto range is not allowed (valid input is from -10 through 10).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).

### Output:

<code>Id</code>	(D_ARRAY_T) Drain-current measured at gate sweep voltage.
<code>Vg</code>	(D_ARRAY_T) Gate-voltage programmed
<code>Gm</code>	(D_ARRAY_T) $Gm=dI_d/dV_g$
<code>Vtx</code>	(double*) $V_{tx}= V_{t0}-V_s/2$
<code>Error:</code>	Error value
	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 91: Series 2400 instruments IdVg

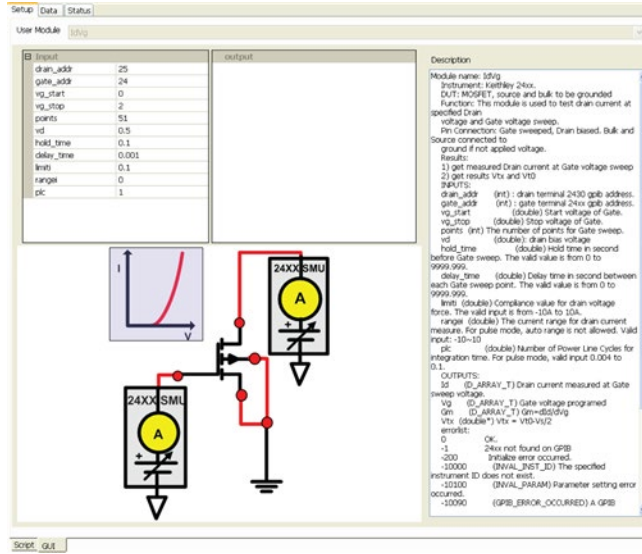
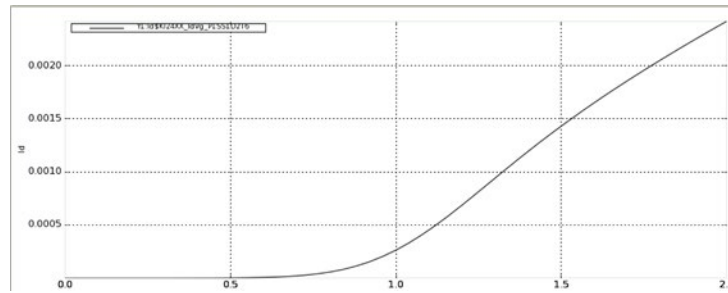


Figure 92: Series 2400 instruments IdVg test results



## 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 Connection: 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  $V_{tx}$  and  $V_{t0}$  results.

### Input:

<code>drain_addr</code> (int):	Model 2430 GPIB drain-terminal address.
<code>gate_addr</code> (int):	Series 2400 SourceMeter GPIB gate-terminal address.
<code>vg_start</code> (double):	Start the gate-voltage pulse (valid input is from -200V to 200V).
<code>vg_stop</code> (double):	Stop the gate-voltage pulse (valid input is from -200V to 200V).
<code>points</code> (int):	Number of drain-sweep points (valid input 1 to 2500).
<code>vd</code> (double):	Drain the bias voltage.
<code>hold_time</code> (double):	Hold time in second before gate sweep. Valid inputs are zero to 9999.999 seconds.
<code>delay_time</code> (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.
<code>limiti</code> (double):	Drain-voltage force compliance value (valid input is from -10A to 10A).
<code>rangei</code> (double):	Range for drain-current measurement. For pulse mode, auto range is not allowed (valid input is from -10 through 10).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).
<code>pulse_width</code> (double):	Duration of the output ON time (valid value is from 0.15ms to 5ms).

## 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.

`pulse_delay` (double): Duration of the output OFF time (valid value is from zero to 9999.999s).

### Output:

<code>Id</code>	(D_ARRAY_T) Drain-current measured at gate sweep voltage.
<code>Vg</code>	(D_ARRAY_T) Gate-voltage programmed
<code>Gm</code>	(D_ARRAY_T) $G_m = dI_d/dV_g$
<code>Vtx</code>	(double*) $V_{tx} = V_{t0} - V_s/2$

- Vt0 (double\*) Calculate  $G_m = dI_d / dV_g$ . Find  $G_{max}$  and extrapolate back to  $I_{ds} = 0$  to find  $V_{t0}$
- 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 93: Model 2430 IdVg Pulse

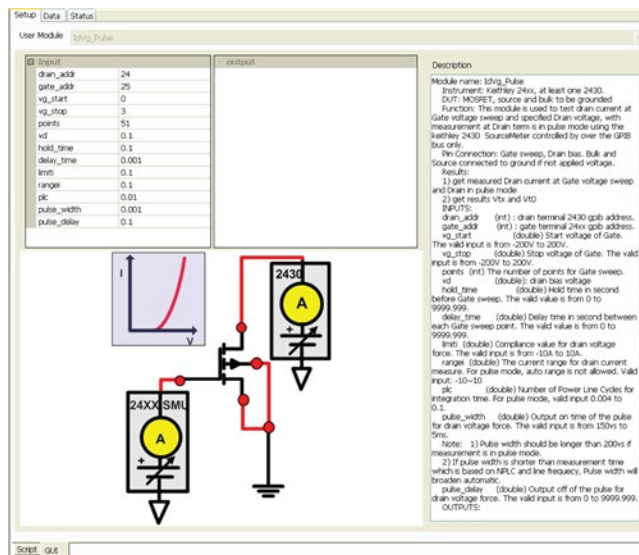
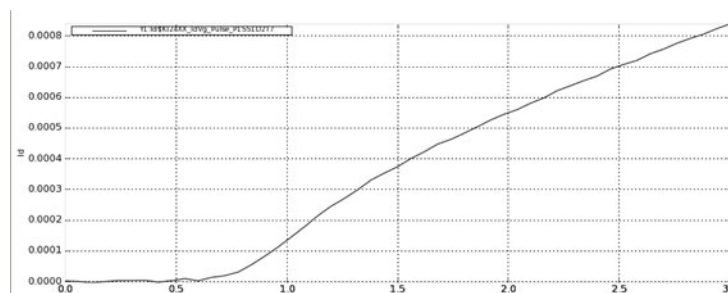


Figure 94: Series 2400 instruments IdVg Pulse test results



## 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 Connection: 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.

### Input:

<code>drain_addr</code> (int):	Model 2430 GPIB drain-terminal address.
<code>gate_addr</code> (int):	Series 2400 SourceMeter GPIB gate-terminal address.
<code>vd_start</code> (double):	Start the voltage-drain pulse.
<code>vd_stop</code> (double):	Stop the voltage-drain pulse.
<code>points</code> (int):	Number of drain-sweep points (valid input 1 to 2500).
<code>limiti</code> (double):	Drain-voltage force compliance value (valid input is from -10A to 10A).
<code>rangei</code> (double):	Range for drain-current measurement. For pulse mode, auto range is not allowed (valid input is from -10 through 10).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).
<code>vg_start</code> (double):	Start the voltage-gate pulse.
<code>vg_stop</code> (double):	Stop the voltage-gate pulse.
<code>vg_step</code> (double):	Step the voltage-gate pulse.
<code>hold_time</code> (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 seconds.
<code>delay_time</code> (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.

### Output:

<code>Vd</code>	(D_ARRAY_T) Drain-voltage programmed.
<code>Id1</code>	(D_ARRAY_T) Drain-current measured at the first gate bias voltage.
<code>Id2</code>	(D_ARRAY_T) Drain-current measured at the second gate bias voltage.
<code>Id3</code>	(D_ARRAY_T) Drain-current measured at the third gate bias voltage.
<code>Id4</code>	(D_ARRAY_T) Drain-current measured at the fourth gate bias voltage.
....	
<code>Error:</code>	Error value
	0 OK.
	-1 Series 2400 SourceMeter instruments 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 95: Series 2400 instruments IdVd

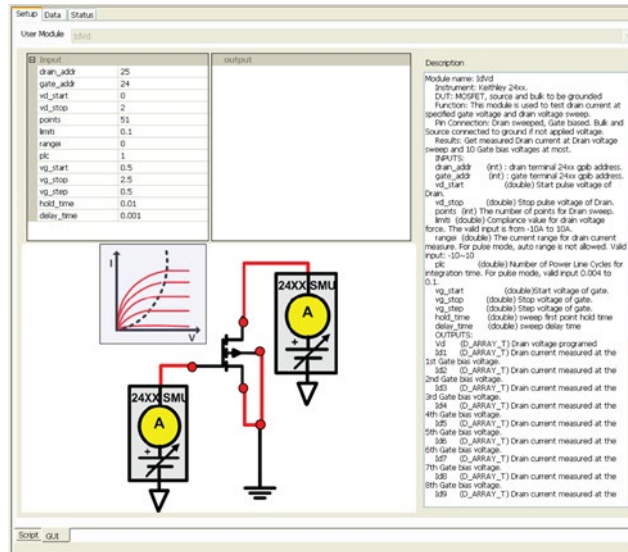
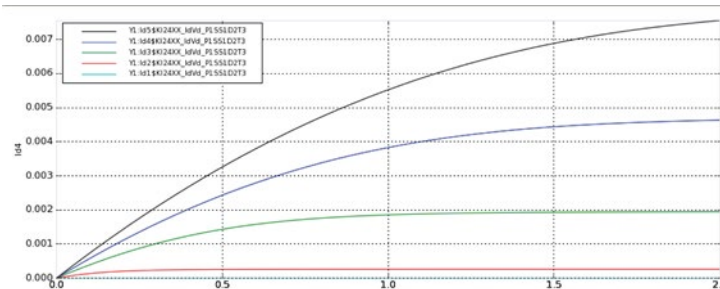


Figure 96: Series 2400 instruments IdVd test results





## 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.

### Input:

<code>drain_addr</code> (int):	Model 2430 GPIB drain-terminal address.
<code>gate_addr</code> (int):	Series 2400 SourceMeter GPIB gate-terminal address.
<code>vd_start</code> (double):	Start the voltage-drain pulse.
<code>vd_stop</code> (double):	Stop the voltage-drain pulse.
<code>points</code> (int):	Number of drain-sweep points (valid input 1 to 2500).
<code>limiti</code> (double):	Drain-voltage force compliance value (valid input is from -10A to 10A).
<code>rangei</code> (double):	Range for drain-current measurement. For pulse mode, auto range is not allowed (valid input is from -10 through 10).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).
<code>vg_start</code> (double):	Start the voltage-gate pulse.
<code>vg_stop</code> (double):	Stop the voltage-gate pulse.
<code>vg_step</code> (double):	Step the voltage-gate pulse.
<code>pulse_width</code> (double):	Duration of the output ON time (valid value is from 0.15ms to 5ms).
<code>pulse_delay</code> (double):	Duration of the output OFF time (valid value is from zero to 9999.999s).

### Output:

<code>Vd</code>	(D_ARRAY_T) Drain-voltage programmed.
<code>Id1</code>	(D_ARRAY_T) Drain-current measured at the first gate bias voltage.

Error: Error value

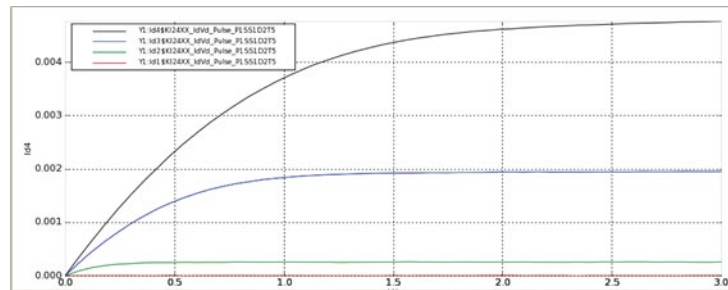
0	OK.
-1	Series 2400 SourceMeter instruments not found on GPIB.
-2	2430 not found on GPIB.
-200	Instrument initialize error.
-300	Configuration error occurred.
-400	Reading error occurred.
-10000	(INVAL_INST_ID) The specified instrument ID does not exist.
-10090	(GPIB_ERROR_OCCURRED) A GPIB communications error occurred.
-10091	(GPIB_TIMEOUT) A timeout occurred during communications.

Figure 97: Series 2400 instruments IdVd pulse

The screenshot displays the ACS software interface for the IdVd pulse test. It is divided into several sections:

- Input Parameters:** A list of parameters including drain\_addr (24), gate\_addr (25), v\_d\_start (0), v\_d\_stop (3), points (51), time (0.1), range (0.1), pc (0.01), v\_g\_start (0.5), v\_g\_stop (2), v\_g\_step (0.5), pulse\_width (0.001), and pulse\_delay (0.1).
- Output:** A blank area for test results.
- Circuit Diagram:** Shows a MOSFET circuit with a 24XX SMU (Source Measure Unit) connected to the gate and drain, and a 2430 current source connected to the drain.
- Description:** A detailed text block explaining the module name (IdVd\_Pulse), instrument (Keithley 24xx), and function (testing drain current with gate and drain voltage sweeps). It lists various parameters like drain\_addr, gate\_addr, v\_d\_start, v\_d\_stop, v\_g\_start, v\_g\_stop, v\_g\_step, pulse\_width, pulse\_delay, pc, and integration time. It also includes a list of return values and error codes such as 0 (OK), -1 (24xx not found on GPIB), -2 (2430 not found on GPIB), -300 (initial error occurred), -301 (configuration error occurred), -400 (reading error occurred), -10000 (INVALID\_ID), and -10100 (INVALID\_PARAM).

Figure 98: 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.

### Input:

<code>gpiб_addr</code> (int):	Instrument GPIB address. Valid input: 1 through 30.
<code>voltage</code> (double):	Pulse level forced.
<code>points</code> (int):	Number of forced pulses (valid input 1 to 2500).
<code>pulse_width</code> (double):	Duration of the output ON time (valid value is from 0.15ms to 5ms).
<code>pulse_delay</code> (double):	Duration of the output OFF time (valid value is from zero to 9999.999s).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).
<code>limiti</code> (double):	Current-sweep compliance value.
<code>rangev</code> (double):	Range for voltage measurement. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.

### Outputs:

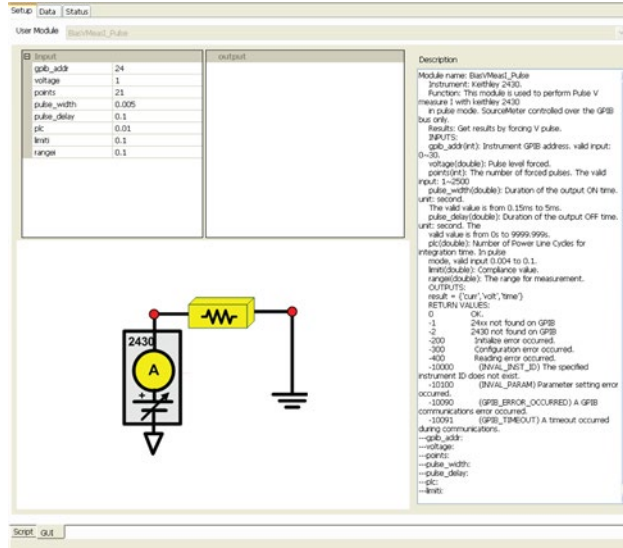
Error:	Error value
	0 OK.
	-1 Series 2400 SourceMeter instruments not found on GPIB.
	-2 2430 not found on GPIB.
	-200 Instrument initialize error.
	-300 Configuration error occurred.
	-400 Reading error occurred.
	-10000 (INVAL_INST_ID) The specified instrument ID does not exist.
	-10090 (GPIB_ERROR_OCCURRED) A GPIB communications error occurred.
	-10091 (GPIB_TIMEOUT) A timeout occurred during communications.

`time`

`I`

`V`

Figure 99: Series 2400 instruments BiasV pulse standard GUI



## Model 2430 current-pulse test

---

Module name: BiasMeasV\_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:

<code>gpiб_addr</code> (int):	Instrument GPIB address. Valid input: 1 through 30.
<code>current</code> (double):	Pulse level forced.
<code>points</code> (int):	Number of forced pulses (valid input 1 to 2500).
<code>pulse_width</code> (double):	Duration of the output ON time (valid value is from 0.15ms to 5ms).
<code>pulse_delay</code> (double):	Duration of the output OFF time (valid value is from zero to 9999.999s).
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs)(valid input 0.004 to 0.10).
<code>limiti</code> (double):	Voltage-sweep compliance value.
<code>rangev</code> (double):	Range for voltage measurement. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.

### Outputs:

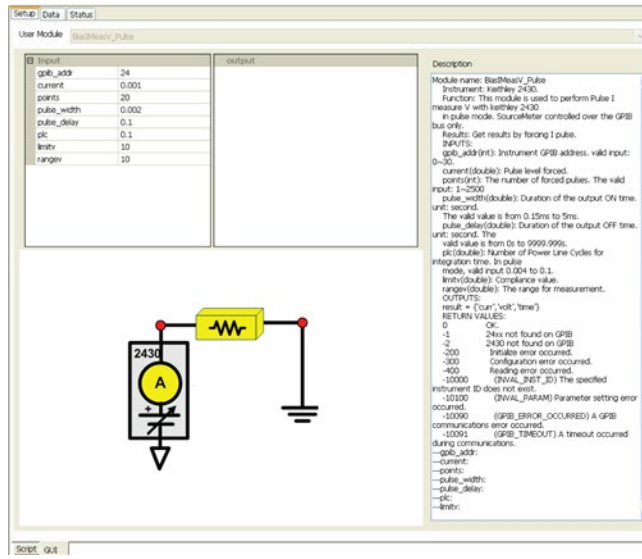
I

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 100: 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 I/V/Time readings for the Models 2400/2410/2420/2425/2430 instruments.

### Inputs:

<code>gpib_addr</code> (int):	Instrument GPIB address. Valid input: 1 through 30.
<code>startv</code> (double):	Start the voltage sweep signal.
<code>stopv</code> (double):	Stop the voltage sweep signal.
<code>points</code> (int):	Number of sweep points (valid input 1 to 2500).
<code>sweepMode</code> (int):	Sweep mode. 0 is a fixed bias. The sampling measurement 1 is linear sweep; 2 is log sweep.
<code>limiti</code> (double):	Current-sweep compliance value.
<code>srangev</code> (double):	Source range for voltage current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.
<code>mrangei</code> (double):	Measurement range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.
<code>hold_time</code> (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 seconds.
<code>delay_time</code> (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.01 to 10).

### Outputs:

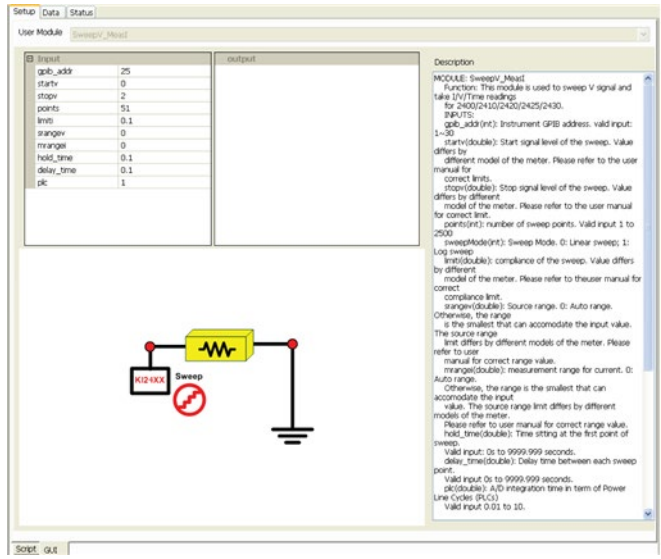
I

V

time

<b>Error:</b>	<b>Error value</b>
	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 101: 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 I/V/Time readings for the Models 2400/2410/2420/2425/2430 instruments.

### Inputs:

<code>gpib_addr</code> (int):	Instrument GPIB address. Valid input: 1 through 30.
<code>starti</code> (double):	Start the current sweep signal.
<code>stopi</code> (double):	Stop the current sweep signal.
<code>points</code> (int):	Number of sweep points (valid input 1 to 2500).
<code>limitv</code> (double):	Voltage-sweep compliance value.
<code>srangei</code> (double):	Source range for current. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.
<code>mrangev</code> (double):	Measurement range for voltage. If zero is selected, the instrument will auto-range. Otherwise, the range is the smallest that can accommodate the input value.
<code>hold_time</code> (double):	Hold time setting at the first sweep point. Valid inputs are zero to 9999.999 seconds.
<code>delay_time</code> (double):	Delay time between each sweep point. Valid inputs are zero to 9999.999 seconds.
<code>plc</code> (double):	A/D integration time in terms of power line cycles (PLCs) (valid input 0.01 to 10).

### Outputs:

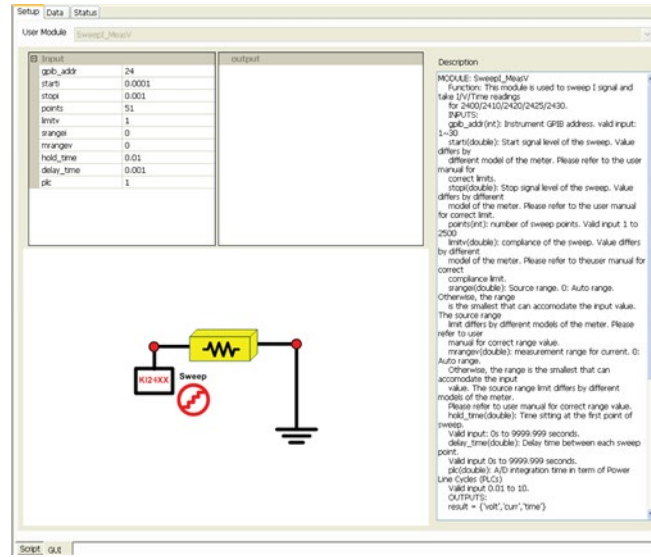
I

V

time

<b>Error:</b>	<b>Error value</b>
	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 102: Series 2400 instruments Sweep standard GUI



---

## UAP and Global variable definitions

### In this section:

---

Overview .....	5-1
Global variables .....	5-2
Global functions .....	5-7
Tools for UAP routines .....	5-23
How to use UAPs .....	5-27

### 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 will describe the global data and global functions that are available within ACS and provide some application examples. All of these descriptions and examples are based on PTMs.

## Global variables

Global data variables can be accessed from a UAP PTM (see the next table). All global variables start with the characters ACS\_XXXX.

Table: ACS global variables

Variable name	Type	Comment	UAP level	Example
ACS_frame	string	The frame of ACS	Any UAP	Object
ACS_uap_level	string	Current UAP level	Any UAP	'wafer_begin', site_end
ACS_proj_path	string	Current path	Any UAP	'C:\ACS\Projects\default'
ACS_proj_name	string	Current project name	Any UAP	'default'
ACS_lot_id	string	Current lot ID	Any UAP except lot begin	'lotid001' (if single wafer)
ACS_slot_no	int	Current slot number	Wafer level and below	1, 2, ...25
ACS_wafer_id	string	Current wafer ID	Wafer level and below	'waferid001'
ACS_pattern_id	string	Current pattern ID	Pattern level and below	'Pattern1'
ACS_site_id	string	Current die ID	Site level and below	'Site_n1p1'
ACS_site_coord	tuple	Current die coordinates	Subsite level and below	(-1, 1)
ACS_sssite_id	string	Current subsite ID	Subsite level and below	'subsite1'
ACS_sssite_loop	int	Current subsite index of subsite loop	Subsite level and below	1
ACS_sssite_loopNum	int	Total loop number of the current subsite	Subsite level and below	1
ACS_sssite_coord	tuple	Current subsite coordinates	Subsite level and below	(0.0,0.0)
ACS_dut_id	string	Current DUT ID	DUT level and below	'Resistor'
ACS_test_id	string	Current test ID	Test level	'sweepv'
ACS_test_data	dictionary	Current test data	Test end level	See the next topic
ACS_wdf_head	dictionary	WDF header information	Wafer level and below	See the next topic
ACS_wdf_nonexist	dictionary	WDF erased sites dictionary	Any UAP	See the next topic
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_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 the next topic
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 current 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_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	Current 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.

Each of these variables can be accessed directly by the name. You can view the variable values by using the logMessage function which is defined in LogManager.py. The messages are viewable in the log window at the bottom of the ACS GUI.

If the variable type is string, you can view its value by 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))
```

Examples are shown below for 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_plp3', (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': '',
'diesizey': 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.

.klf head Structure:

```
{
  "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
}
```

.klf limits structure:

```
[{
  "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))
```

- Usage of ACS\_Stress\_Time/ ACS\_Stress\_Duration/ ACS\_Disable\_Subsite\_List/ ACS\_Disable\_DUT\_List/ ACS\_Disable\_Test\_List - you can find the DC stress-measure loop project in the following location: (C:\ACS\Projects\ DC\_Stress\_Measure\_loop) and Pulse stress-measure loop project (C:\ACS\Projects\ Pulse\_Stress\_Measure\_Loop). Note that the Pulse stress-measure loop project is only available when ACS is installed on the Model 4200-SCS.

## Global functions

Functions that can be accessed in the UAP PTMs are listed in the next Table:

Table: ACS global functions

Function type	Function name	Comment
User Global Data Functions	FUNC_SET_GLOBAL_VALUE (name, value)	Update global variable with a value. Only this function can change a global variable value.
	FUNC_GET_GLOBAL_VALUE (name)	Get a global variable's value
	FUNC_SET_USER_GLOBAL	Define new variable in user global data pool. Only used between UAP
	FUNC_SYS_GLOBAL (name, value)	Set system global variable value
Execution Engine Related Functions	FUNC_EXIT_KTXE	Exit ktxe execution
	FUNC_SET_KTXE_SUBSITES	Set subsites list used by the execution engine
	FUNC_SET_KTXE_LOOP_WAFER	Set wafer loop flag
	FUNC_EXEC_TEST	Execute the specified test module
KDF Manipulation Related Functions	FUNC_GET_KDF_HEADER	Get kdf header; returns data as a dictionary
	FUNC_SAVE_KDF_HEADER	Save kdf header dictionary into a kdf file
	FUNC_SAVE_KDF_WAFERID	Save kdf wafer id into a kdf file
	FUNC_SAVE_KDF_EOW	Save kdf wafer delimiter (<EOW>) into a kdf file
	FUNC_SAVE_KDF_SITEID	Save kdf site id into a kdf file
	FUNC_SAVE_KDF_EOS	Save kdf site delimiter (<EOS>) into a kdf file
	FUNC_GET_KDF_DATA	Get kdf data for the specified wafer and site id
	FUNC_GET_NEW_KDF_DATA	Get the latest kdf data for the specified wafer and site id
	FUNC_SAVE_KDF_DATA	Save kdf data into a kdf file
	FUNC_SITE_COORD_2_ID	Translate site coordinate to site id
FUNC_SITE_ID_2_COORD	Translate site id to site coordinate	
Get Object Functions	FUNC_GET_WDF_OBJ	Get the wdf object used by the execution engine
	FUNC_GET_KDF_OBJ	Get kdf object from the test tree used by the execution engine

	FUNC_GET_SUBSITE_OBJ	Get the subsite object from the test tree used by the execution engine according to the subsite name
	FUNC_GET_DUT_OBJ	Get the DUT object from the test tree used by the execution engine according to the DUT's name
	FUNC_GET_TEST_OBJ	Get the test object from the test tree according to the test name
Get Test Info Functions	GET_EXEC_TEST_DIC	Get content of the test dictionary
	GET_EXEC_TEST_LIST	Get test list. It is the key in test dictionary
	FUNC_GET_DUT_NAME	Get the DUT's name

## FUNC\_SET\_GLOBAL\_VALUE (name, value)

Function: Update the global variable with a value. Only this function can be used to change a global variable's value.

Parameter:

Name: string; global variables are listed in Table 1

Value: string

Example:

```
FUNC_SET_GLOBAL_VALUE('ACS_wafer_id', '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 the .kdf file. If you want the updated value stored in the .kdf file, use the KDF Manipulation Related Functions.

## FUNC\_GET\_GLOBAL\_VALUE (name)

Function: Get global variable's value. You can also use the variable's name directly.

Parameter:

Name: string, variables which are listed in Table 1

Return: string

Example:

```
FUNC_GET_GLOBAL_VALUE('ACS_wafer_id')
Or ACS_wafer_id
```

## FUNC\_SET\_USER\_GLOBAL (name)

Function: Add a global user variable to the data pool. After the variable is set, this user global variable can be used by name directly at current UAP and below.

Parameter:

Name: string

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 (value)

Function: Set the ACS\_ktxe\_loop\_wafer global variable which is the current wafer loop flag. If the value is 1, then the current wafer will be tested repeatedly until the flag is set to 0. This function is only effective in single wafer mode.

Parameter:

Value: int. 1 or 0

Example:

```
FUNC_SET_KTXE_LOOP_WAFER(1)
```

## FUNC\_SYS\_GLOBAL (name, value)

Function: Set the system global variable's value.

Parameter:

Name: string, global variables which are listed in Table 1

Value: string

Example:

```
FUNC_SYS_GLOBAL('ACS_wafer_id', 'wafer_1')
```

## FUNC\_EXIT\_KTXE ()

Function: Exit the execution engine.

Example:

```
FUNC_EXIT_KTXE()
```

## FUNC\_SET\_KTXE\_SUBSITES (subsite list)

Function: Change the subsite list in the execution engine. Used on the site level UAP and below.

Parameter:

subsitelist: list, in format of [(x1,y1), (x2, y2), ...]

Example:

```
FUNC_SET_KTXE_SUBSITES([(0.0,0.1),(0.2,0.3)])
```

## FUNC\_GET\_KDF\_HEADER ()

Function: Get the kdf header.

Return: dictionary; If a kdf exists. If a kdf does not exist = False.

Example: Use this function at the beginning of the UAP lot.

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 data:

```
{
  '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\_HEADER (kdf\_file, header)

Function: Save kdf header dictionary into a corresponding kdf file.

Parameter:

kdf\_file: string, the full name of kdf file.

header: dictionary, kdf header information.

Return:

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)

Function: Save kdf wafer id into a corresponding kdf file.

Parameter:

kdf\_file: string, the full name of kdf file.

wafer\_id: string, ID of wafer

Return:

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)

Function: Save kdf wafer delimiter (<EOW>) into a kdf file.

Parameter:

kdf\_file: string, the full name of kdf file.

Return:

True: If write to file was successful.

False: If write to file was not successful.

Example:

```
FUNC_SAVE_KDF_EOW('C://a.kdf')
```

## FUNC\_SAVE\_KDF\_SITEID (kdf\_file, site\_id, site\_x, site\_y)

Function: Save kdf site id into a kdf file

Parameter:

kdf\_file: string, the full name of kdf file

site\_id: string, ID of site

site\_x: string, x-coordinate of the site

site\_y: string, y-coordinate of the site

Return:

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')
```

## FUNC\_SAVE\_KDF\_EOS (kdf\_file)

Function: Save kdf site delimiter (<EOS>) into a kdf file.

Parameter:

kdf\_file: string, the full name of kdf file

Return:

True: If write to file was successful.

False: If write to file was not successful.

Example:

```
FUNC_SAVE_KDF_EOS('c:\\a.kdf')
```

## FUNC\_SAVE\_KDF\_DATA (kdf\_file, data)

Function: Save kdf data into a kdf file.

Parameter:

kdf\_file: string, the full name of kdf file data: list, the data which is to be written into the kdf file

Return:

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)

Function: Get the kdf data for the specified wafer and site id

Parameter:

wafer\_id: string, ID of wafer.

site\_id: string, ID of site.

Return:

List of data: If the kdf data file was read successfully.

False: If the read from the kdf data file was not successful.

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)))
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)

Function: Get the latest kdf data for the specified wafer and site id. It differs from the function FUNC\_GET\_KDF\_DATA (wafer\_id, site\_id) only when the site sequence priority is pattern first and the same site is selected in multiple patterns.

Parameter:

wafer\_id: string, ID of wafer.

site\_id: string, ID of site.

Return:

list: If the read of the kdf data was successful.

False: If the read from kdf file was not successful.

Return Structure:

```
[[padname@testname@subsitename#subsitlistid@groupname],...]
```

Example:

Build a project with two patterns and select the same site in both patterns. Work in single mode. Execute the following code at site end UAP for an example of how to use this function.

```
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', 0.012356422404429968]]
Line OUTPUT:
[['I_Pos@itm_2@HOME_2#1@GROUP1', 0.36503035785604399],
['I_Pos@itm_2@HOME_2#2@GROUP1', 0.012356422404429968]]
```

## FUNC\_SITE\_COORD\_2\_ID (coord)

Function: Translate the site coordinate to site id.

Parameter:

coord: tuple, site coordinates

Return:

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))))
```



## FUNC\_SITE\_ID\_2\_COORD (site\_id)

Function: Translate the site id to site coordinate.

Parameter:

siteid: string

Return:

coord: tuple, site coordinates. : In 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)))
```

## FUNC\_GET\_WDF\_OBJ ()

Function: Get the wdf object used by the execution engine. Use this wdf object to get properties and to be able to call functions. All properties and functions can be called by '.' operator after getting the wdf object

WDF object Property:

```
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 Operation:

```
load(wdf_file): load .wdf file into a wdf object
save(wdf_file): save wdf object to .wdf file
clear(): clear wdf object.
reset(): reset wdf object's all properties to default values.
```

Parameter

Return:

wdf\_object : object, if wdf object is in the execution engine.

None: if no wdf object in execution engine.

Example:

```
import LogManager as log
log.logMessage(str(FUNC_GET_WDF_OBJ().head))
FUNC_GET_WDF_OBJ().clear()
```

## GET\_EXEC\_TEST\_DIC ()

Function: Get the contents of the test dictionary.

Parameter:

Return:

test\_dict: dictionary, site coordinates. : In form of (1,-2)

None: If failure to extract contents.

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()))  
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 (location)

Function: Execute a test.

Parameter:

location: tuple, with format of (waferid, pattern\_name, site\_id, subsite\_name, device\_name, test\_name)

Return:

test data: dictionary,

None: If execution was unsuccessful.

Return Structure:

Example:

```
import LogManager as log  
log.logMessage(str(FUNC_EXEC_TEST(('03', 'Pattern_1', 'Site_n2p2', 'subsite',  
 'Resistor_1k', 'sweepv'))))  
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 ()

Function: Get the execution test list.

Parameter:

Return:

test\_list: list

None: If fail.

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()))
```

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)

Function: Get the test object from the test tree used by the execution engine.

Parameter:

test\_name: string, ID of test

Return:

test\_object: Object

None: If fail.

Test Object property:

```
self.msg
self.commonSetting
self.SMU
self.outputs
self.limit
self.dut
```

Test Object function:

```
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': 9.999999999999997e+098,
'ConsFail': 0,
'ValidLow': -9.999999999999997e+098,
'Critical': 0,
'Exit': 'None',
'SpecLow': -9.999999999999997e+098,
'Report': 1,
'Sigma': 1.0,
'Unit': 'A',
'SpecHigh': 9.999999999999997e+098}, ...]
```

## FUNC\_GET\_SUBSITE\_OBJ (subsite\_name)

Function: Get the subsite object from the test tree used by the execution engine.

Parameter:

subsite\_name: string, ID of subsite

Return:

subsite\_object: Object

None: If fail.

Subsite Object property:

```
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)

Function: Get the DUT object from the test tree used by the execution engine. Use the DUT object to get properties and to be able to call functions

DUT object property:

```
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 function:

```
getSMUInfo(subDevNo=-1):Get SMU information according to subdevice No.
reset(): Reset SMU information to default setting
```

Parameter:

**dut\_name:** string

**Return:**

**dut\_object:** If success

**None:** If fail.

**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)

Function: Get the DUT's name according to the test name. Same value as ACS\_dut\_id

Parameter:

test\_name: string

Return:

dut\_name: string, If successful

None: If 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)

Function: Get the kdf object from the test tree used by the execution engine.

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

### Importing python modules

Defined modules can be reused by another program by using the import command. Here are the ways to import a module.

- `import X`: imports the module X. After you've run this statement, you can use `X.name` to refer to things defined in module X.
- `import X as Y`: just rename module X as Y. When X is a long name, we will use a short alias Y instead.
- `from X import *`: imports all public objects from the module X. This would allow you to simply use name to refer to things defined in module X. X itself is not defined, so `X.name` is not valid. If name was already defined, it is replaced by the newer version. If name in X is changed to point to some other object, your module won't notice. So we strongly suggest NOT using this style of import.

`from X import a, b, c`: imports a,b,c objects from the module X. You can now use a, and 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 UAP routine using the import statement. For example, `import LogManager`.

- **LogManager:** This module provides functions to print information to log window, there are six functions to print information according to log level.

```
logMessage(message)
logDebug(message)
logInfo(message)
logWarning(message)
logError(message)
logCritical(message,color=False): when color is True, message will be printed in red.
```

- **.klf:** This 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 ACS Reference manual (document number: ACS-901-01) for more information. The `.klf` functions are provided as an example (see next Example).

Example `.klf` functions:

```
KLF.KLF(klf): Construct klf object. .klf file is loaded into the KLF object.
.load(klf): Load a KLF file.
.clear(): Clear the KLF object. head is set to {}, limits is set to [],and msg is set to None.
.save(klf, limits): Save limit list into a KLF file.
.find(ID): Search ID through the limit list, return reference of the item if found.
.remove(ID): Remove an item from the limit list according to the ID given.
.new(ID = ""): Create a new limit item with all default values.
.reset(ID_or_lim): Reset a limit item with default values, its ID is reserved.
.update(ID_or_lim, **attrib): Update a limit item with the new attributes given.
.insert(index, ID_or_lim): Insert an item at the specified index of the limit list.
.append(ID_or_lim): Append an item at the end of the limit list.
.validate(ID_or_lim, autofix = False): Validate a limit item.
.test(ID, value): Test whether a parameter is in spec, valid, or invalid.
```

- **kimisc:** This module provides some common operations used in ACS. Functions are showed as below.

```
atoi(str, base = 10) : Convert a string into integer using the optional base(default is decimal). The difference between this function and Python standards string.atoi(x[,base]) is, this function allows you to input a string with characters other than decimal digits. atoi attempts to parse as many characters as it can to build an integer.
atof(val): Convert a string into float. The difference between this function and Python standards 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): operation for ini file,
    Extract 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
:
:
putSetting(filename, section, item, value, openc = '[', closec = ']', asignc =
 '='): an operation for ini file, corresponding to getSetting, put the section's
 item value.
delSetting(filename, section, item, openc = '[', closec = ']', asignc = '='): an
 operation for ini file, delete the section's item.
avgsdev(data): Get average and standard deviation of data.
    if data's type is float or int :return(data,0) if data's type is list and
    len(data)=1: return (data[0],0)
        if data is float list, return [average,standard deviation]
copy_tree(src, dest): copy directory from src to dest.
getfoldersize(dir): compute directory's size, unit as KB.

```

## Modules in python

### NOTE

All of python's standard and extension modules can be used in a UAP. If you want to use these modules, you must import them to a UAP routine using the import statement. For example, `import scipy`.

Also, all python modules can use the `help()` function to get the module's or function's documentation in a PythonWin interactive window. The next Figure is an example.

**Figure 103: Sample help() description**

```

PythonWin - [Interactive Window]
File Edit View Tools Window Help
PythonWin 2.4.3 (#68, Apr 11 2006, 15:32:42) [MSC v.1310 32 bit (Intel)] on win32
Portions Copyright 1994-2004 Mark Hammond (mhammond@skippinet.com.au) - see 'Help/About PythonWin' for further copyright information.
>>> import scipy
>>> help(scipy.arange)
Help on built-in function arange in module numpy.core.multiarray:

arange(...)
    arange([start,] stop[, step], dtype=None)

    For integer arguments, just like range() except it returns an array
    whose type can be specified by the keyword argument dtype.  If dtype
    is not specified, the type of the result is deduced from the type of
    the arguments.

    For floating point arguments, the length of the result is ceil((stop -
    start)/step).  This rule may result in the last element of the result
    being greater than stop.

>>> help(scipy)
Help on package scipy:

NAME
    scipy

```

- **copy**

```
deepcopy(obj): Deep copy operation on arbitrary Python objects. Mostly used with recursive objects(object contains another object)
```

- **numpy**

```
sin(): get sin value
```

```
pi: const pi
```

```
array: Return an array from object with the specified data-type.
```

```
sum: Sum the array over the given axis.
```

- **os**

```
os.path module provides many useful common pathname manipulations. Please see help(os.path) for details in a PythonWin window.
```

```
os.path.join(a, *p): Join two or more pathname components, inserting "\" as needed.
```

```
os.mkdir(path): Create a directory. If an intermediate directory doesn't exist, an error is raised.
```

```
os.makedirs(path): Super-mkdir. Recursive create a directory. If a directory doesn't exist, create it.
```

- **scipy**

```
min, max, median, average, stddev
```

- **shutil**

```
shutil.copy2(srcname, destname): Copy data and all state info
```

- **string**

```
string.atof(s): Return the floating point number represented by the string s.
```

- **time**

```
time.sleep(t): Delay execution for a given number of seconds.
```

```
time.strftime (format[, tuple]): Return format string. For example:
```

```
time.strftime("%Y.%m.%d %H:%M:%S", time.localtime())
```

- **types**: Define names for all type symbols known in the standard interpreter. For additional detail, use the help(types) in PythonWin after you import types.
- **xls**: Excel module with xml format. For additional detail, use the help(xls) in PythonWin after you import xls

## Modules in wxpython

- wx.MessageDialog(): Popup message window
- wx.TextEntryDialog: Popup text entry dialog, get the input value
- wx.grid

## .dll modules

- KIGPIB\_KATS

## File operation

Refer to the help (file) description in PythonWin. Also, refer to help (os.path) to get information regarding common pathname manipulation routines.

- file(name[, mode[, buffering]]): Open a file. The mode can be 'r', 'w' or 'a' for reading (default), writing or appending.
- open(): an alias for file().
- write(str): Write string str to file.
- read([size]): read at most size bytes, returned as a string.
- close(): Close the file

### NOTE

All library and module names are case sensitive.

## How to use UAPs

UAP routines are used for two purposes:

- Monitor the test process
- Write a user data file

The following describes how to use a UAP in these two different conditions.

## Control test process

You can change the normal test process by changing some global variable flags according to a certain condition. For example, FUNC\_SET\_KTXE\_LOOP\_WAFER() and FUNC\_EXIT\_KTXE(). You can also halt the execution process by popping up a text input dialog or message dialog under some conditions. There are three example files, ChooseFileToSave.py OKCancelDialog.py and QueryEntryDlg.py in pylibray, that show how to call a wxpython dialog from a UAP routine.

## 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 UAP routine. Refer to the code in ChooseFileToSave.py or the site\_ACS\_to\_file.py for additional details.

Specifications are subject to change without notice.  
All Keithley trademarks and trade names are the property of Keithley Instruments.  
All other trademarks and trade names are the property of their respective companies.

Keithley Instruments  
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-800-935-5595 • [www.keithley.com](http://www.keithley.com)

---



A Greater Measure of Confidence