Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with non-hazardous 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 rated Measurement Category I and Measurement Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Measurement Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Measurement Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the user documentation.

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 30V RMS, 42.4V peak, or 60VDC 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 1000V, 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.

The instrument and accessories must be used in accordance with its specifications and operating instructions, or the safety of 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 safety earth 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 \[\text{\text{-}}\] screw is present, connect it to safety earth ground using the wire recommended in the user documentation.

The \[\text{\text{-}}\] symbol on an instrument means caution, risk of danger. The user should refer to the operating instructions located in the user documentation in all cases where the symbol is marked on the instrument.

The \[\text{\text{-}}\] symbol on an instrument means caution, risk of danger. Use standard safety precautions to avoid personal contact with these voltages.

The \[\text{\text{-}}\] symbol on an instrument shows that the surface may be hot. Avoid personal contact to prevent burns.

The \[\text{\text{-}}\] symbol indicates a connection terminal to the equipment frame.

If this \[\text{\text{-}}\] 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.
# Table of Contents

## ACS Programming Overview

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

## LPT Command Reference

- LPT command introduction .................................................................................................................. 2-1
- TSP LPT commands .............................................................................................................................. 2-1
  - TSP LPT Library Command Examples ................................................................................................. 2-17
- Python LPT library .............................................................................................................................. 2-20
  - Python LPT introduction .................................................................................................................. 2-20
  - Python LPT functions ...................................................................................................................... 2-21
  - ACS LPT library commands ........................................................................................................... 2-23
- PTM examples ....................................................................................................................................... 2-56

## Series 2600A Library and Python Library

- Series 2600A library introduction ....................................................................................................... 3-1
  - Series 2600A general notes ............................................................................................................. 3-2
- Device library ....................................................................................................................................... 3-3
  - BJT library ....................................................................................................................................... 3-3
  - MOSFET library ............................................................................................................................... 3-12
  - Diode library .................................................................................................................................... 3-21
  - Resistor library .................................................................................................................................. 3-23
- WLR library overview .......................................................................................................................... 3-24
  - HCI .................................................................................................................................................. 3-24
  - TDDDB_CCS ..................................................................................................................................... 3-29
  - TDDDB_per_pin ................................................................................................................................. 3-32
  - NBTI .................................................................................................................................................. 3-35
  - NBTI_meas .......................................................................................................................................... 3-37
  - NBTI_on_the_fly ............................................................................................................................... 3-43
  - qbd_rmpj .......................................................................................................................................... 3-45
  - qbd_rmpv .......................................................................................................................................... 3-48
  - Em_iso_test ...................................................................................................................................... 3-52
- Python user library introduction ......................................................................................................... 3-56
  - Configure a capacitor meter library ................................................................................................. 3-56
  - Configure a switch matrix library ..................................................................................................... 3-62
  - Configure a scope library .................................................................................................................. 3-68
  - Configure a Series 23x library ........................................................................................................... 3-69
  - Configure a Series 3700 system switch DMM library ....................................................................... 3-78
  - Configure a Series 2400 SourceMeter instruments library ............................................................... 3-79

## Python Test Module (PTM) Debug Tool

- PTM debug tool introduction ............................................................................................................... 4-1
- PythonWin description ......................................................................................................................... 4-1
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTM debugging</td>
<td>4-4</td>
</tr>
<tr>
<td>Start debugging</td>
<td>4-6</td>
</tr>
<tr>
<td>Debug tool limitations</td>
<td>4-19</td>
</tr>
<tr>
<td><strong>UAP and Global Variable Definitions</strong></td>
<td>5-1</td>
</tr>
<tr>
<td>Overview</td>
<td>5-1</td>
</tr>
<tr>
<td>Global variables</td>
<td>5-1</td>
</tr>
<tr>
<td>Global functions</td>
<td>5-10</td>
</tr>
<tr>
<td>Useful tools for UAP routines</td>
<td>5-30</td>
</tr>
<tr>
<td>Importing python modules</td>
<td>5-30</td>
</tr>
<tr>
<td>Useful modules in ACS</td>
<td>5-31</td>
</tr>
<tr>
<td>Useful modules in python</td>
<td>5-33</td>
</tr>
<tr>
<td>Useful modules in wxpython</td>
<td>5-34</td>
</tr>
<tr>
<td>Useful .dll module</td>
<td>5-34</td>
</tr>
<tr>
<td>File operation</td>
<td>5-35</td>
</tr>
<tr>
<td>How to use UAPs</td>
<td>5-35</td>
</tr>
<tr>
<td>Control test process</td>
<td>5-35</td>
</tr>
<tr>
<td>Write user data file</td>
<td>5-35</td>
</tr>
</tbody>
</table>
ACS Programming Overview

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 2600A-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 2600A-series or 700-series) instrumentation. An added feature of creating PTM tests, is the ability to create test modules with customer Graphical User Interfaces.

- 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) CTM 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 will cover using the Keithley LPTLib instrument functions as well as introduce the user to creating 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 such as those listed below:

- [http://www.lua.org/manual](http://www.lua.org/manual)
- Programming in Lua, 2nd Edition, Roberto Ierusalimschy, Lua.org (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

<table>
<thead>
<tr>
<th>Tools</th>
<th>Help</th>
</tr>
</thead>
<tbody>
<tr>
<td>KYLT</td>
<td></td>
</tr>
<tr>
<td>KCON</td>
<td></td>
</tr>
<tr>
<td>Test Script Builder</td>
<td></td>
</tr>
<tr>
<td>Preferences</td>
<td></td>
</tr>
<tr>
<td>User Accounts</td>
<td></td>
</tr>
<tr>
<td>Read Current Test Data</td>
<td></td>
</tr>
<tr>
<td>Configure Hardware</td>
<td></td>
</tr>
<tr>
<td>Firmware Refresh Kit</td>
<td></td>
</tr>
<tr>
<td>Offline Data Plotting</td>
<td></td>
</tr>
<tr>
<td>Custom Test GUI Designer (XRC)</td>
<td></td>
</tr>
<tr>
<td>Script Editor</td>
<td></td>
</tr>
<tr>
<td>Convert Data</td>
<td></td>
</tr>
<tr>
<td>Graphically Define a New Device</td>
<td></td>
</tr>
</tbody>
</table>
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**

![Parameter tab screenshot](image.png)

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**

<table>
<thead>
<tr>
<th>Name</th>
<th>input/output</th>
<th>Data Type</th>
<th>Default</th>
<th>Min/valid tuple</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpin</td>
<td>input</td>
<td>int</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lpin</td>
<td>input</td>
<td>int</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vforce</td>
<td>input</td>
<td>int</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>resistance</td>
<td>output</td>
<td>float</td>
<td>resistance</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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**

```python
from ACSLPT import *
from ACS_PostData import *
from ptinlpt.constlpt import *
```
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**

```python
def resv(hpin=0, lpin=0, vforce=0, resistance='resistance'):
    openpin(SMU, hpin)  #Open the instruments and connect the high terminal
    addpin(SMU, GND)    #Connect the low terminal to ground
    forcev(SMU, vforce) #Force voltage
    temp_current = measi(SMU) #Measure current
    devint()           #Clear the instruments and connections
    temp_resistance = vforce/temp_current #Calculate resistance
    ACSPostDataDouble("resistance", temp_resistance) #Return the data to the sheet
```

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Imports</th>
<th>Module Description</th>
<th>Log Console</th>
</tr>
</thead>
</table>

| 2011-02-26 12:29:12 | PTM library save completed, the path is: C:\ACS\library\pyLibrary\PTMLib\eTest.py |

| 2011-02-26 12:31:19 | Checking completed, No error found |


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: 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 command introduction

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

The Keithley Instruments Automated Characterization Suite (ACS) incorporates two LPT libraries. One of the LPT libraries contains commands that are generally used when creating ACS Script Test Modules (STM) and can be used to control Keithley Instruments Series 2600A System SourceMeters®, Model 707B/708B switch matrices, or Model 37xx instruments. This library is called the STM LPT Library.

The second library contains commands that are generally used when creating ACS Python Test Modules (PTM) and can be used with the Model 4200-SCS, Series 2600A System SourceMeters®, Series 2400 instruments and Series 23X instruments. In addition, the Python LPT library contains commands that allow you to control other GPIB-based instruments.

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

TSP LPT commands

This section describes the functions in the ACS TSP LPT Library. The TSP LPT library is used when creating STMs.

NOTE
The LPT commands are listed in alphabetical order.

addcon

Purpose: Add terminal-pin connections after an initial connection has been made.

Format:
```
addcon(*instMTRX, ter, pin, *more_pin)
```
**inst MTRX** — the matrix name in the hardware configuration; it’s optional.

**Ter** — list of terminals to connect.

**Pin** — list of pins to connect.

**more pin** — more pins to connect.

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

**addcon** and **conpin** commands can be used together. **addcon** differs from **conpin** in that **addcon** does not clear previous connections nor does it clear the instruments.

**addcon** supports both row-column mode (for example, instruments are connected to the row connectors of the matrix card and pins are connected to the column connectors of the matrix card) and instrument-card mode (instruments and pins are connected to matrix card columns and the rows are assigned automatically to connect the pins and instruments).

For more information about row-column and instrument card modes, refer to the ACS Reference Manual (document number: ACS-901-01).

Examples:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addcon MTRX1,SMU1,1</code></td>
<td>Connection explicitly specifying the matrix</td>
</tr>
<tr>
<td><code>addcon(SMU1,1)</code></td>
<td>Connect SMU1 to pin 1</td>
</tr>
<tr>
<td><code>addcon(SMU1H,1)</code></td>
<td>Connect the high terminal of SMU 1 to pin 1</td>
</tr>
<tr>
<td><code>addcon(SMU1L,1)</code></td>
<td>Connect the low terminal of SMU1 to pin 1</td>
</tr>
<tr>
<td><code>addcon(SMU1,1,2,3)</code></td>
<td>Connect SMU1 to pins 1, 2, 3</td>
</tr>
<tr>
<td><code>addcon({SMU1,SMU2}, {1,2})</code></td>
<td>Connect SMU1 to pin 1, connect SMU2 to pin 2</td>
</tr>
</tbody>
</table>

**addconrc**

**NOTE**

The **addconrc** command cannot be mixed with **conpin** and **addcon** command statements.

Purpose: Explicitly connect the rows and columns of the matrix.

Format:

```
addconrc(unitname, row_list, col_list)
```

**unitname** — the matrix name in hardware configuration.

**row_list** — list of rows to connect.

**col_list** — list of columns to connect.

Remarks: **addconrc** is generally used when the switch matrix is configured in row-column mode. For more information about row-column and instrument card modes, refer to the ACS Reference Manual (document number: ACS-901-01).
Example: 70X

```
addconrc(MTRX1, 'A',1)   -- Connect MTRX1 Row A to Col 1
addconrc(MTRX1, {'A','B'}, {1,2}) -- Connect MTRX1 Row A to Col 1 and Row B to Col 2
```

Series 3700 System Switch

```
addconrc(MTRX1, '1',1)   -- Connect Row 1 to Col 1
addconrc(MTRX1, {'1','2'}, {1,2}) -- Connect Row 1 to Col 1, and Row 2 to Col 2
```

addpth

Purpose: Add terminal-pin connections by path (row).

Format:

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

*inst MTRX — the matrix name in hardware configuration; it's optional.

Ter — list of terminals to connect.

Pin — list of pins to connect.

Row — the row used to connect terminals and pins.

Example: 70X

```
addpth(MTRX1,SMU1,1,'A')  -- Using MTRX1, connect SMU1 to pin 1 using Row A
addpth(SMU1,1,'A')   -- Connect SMU1 to pin 1 via Row A
addpth(SMU1H,1,'A')   -- Connect the high terminal of SMU1 to pin 1 using Row A
addpth(SMU1L,1,'A')   -- Connect the low terminal of SMU1 to pin 1 using Row A
addpth({SMU1,SMU2},{1,2},'A') -- Connect SMU1 to pin 1 using row A, and SMU2 to pin 2 using row B
addpth(MTRX2, {SMU1,SMU2},{},'A') -- On MTRX2, Connect SMU1 to SMU2 using Row A
addpth({}, {1,2}, 'A')   -- Connect pins 1 & 2 together using Row 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: The addpth command requires you to specify a matrix row (or path) to use. This command cannot be used to connect paths across multiple matrices.

In addition, you can only connect terminals or pins with this function. However, when only connecting terminals, the instMTRX argument is required; otherwise, errors will result.
For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference Manual (document number: ACS-901-01).

**clrcon**

Purpose: Clears all matrix connections.

Format:

```
clrcon(unitname)
```

*unitname* – The instrument name in `\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`. For example, *MTRX1*.

Examples:

```
clrcon()   -- Clear all connections on all matrices.
clrcon(MTRX1)  -- Clear the connections on MTRX1 only.
```

**Figure 17: clrcon library command**

---

**avgi/avgv**

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

Format:

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

*Itable* – User-created variable or array; the measured current value is saved to Itable[1].

*Vtable* – User-created variable or array; the measured voltage value is saved to Vtable[1].

*points* – The number of measurement points to average. This number ranges from 1 to 160,000.

*step_time* – The interval in seconds between each measurement points. Minimum practical time is approximately 0.0001s (nplc must be set as 0.001 to achieve the minimum practical time).

**clrscn**

Purpose: Clears the measurement scan tables associated with a sweep.
Format:
```c
clrscn()
```

### conpin

Purpose: Clears all sources and connections and makes new terminal-pin connections.

Format:
```c
conpin(*instMTRX, ter, pin, *more_pin)
```

- `instMTRX` – the matrix name in hardware configuration; it's optional.
- `ter` – list of terminals to connect.
- `pin` – list of pins to connect.
- `more_pin` – more pins to connect

Remarks: `conpin` supports both row-column mode and instrument-card mode of the matrix. When the matrix is set to instrument-card mode, the rows will be assigned automatically.

For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference Manual (document number: ACS-901-01).

Example:
```c
conpin(MTRX1,SMU1,1)  -- On MTRX1, connect SMU1 to pin 1
conpin(SMU1,1)   -- Connect SMU1 to pin 1
conpin(SMU1H,1)   -- Connect the high terminal of SMU1 to pin 1
conpin(SMU1L,1)   -- Connect the low terminal of SMU1 to pin 1
conpin(SMU1,1,2,3)  -- Connect SMU1 to pins 1, 2, and 3
conpin({SMU1,SMU2},{1,2}) -- Connect SMU1 to pin 1, and SMU2 to pin 2
```

### conpth

Purpose: Clears all the current connections and makes new terminal-pin connections by path.

Format:
```c
conpth(*instMTRX, ter, pin, row)
```

- `instMTRX` – the matrix name in hardware configuration; this argument is optional.
- `ter` – list of terminals to connect.
- `pin` – list of pins to connect.
- `row` – the row used to connect terminals and pins.

Remarks: All terminals and pins will be connected together using the specified row.

A single command cannot be used to connect two paths.
A single command cannot be used to connect paths in multiple matrices.

This command can only be used to connect pins or terminals connect terminals. However, when connecting terminals, the instMTRX argument is required, otherwise errors will result.

**Example: 70X**

```plaintext
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**

```plaintext
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')
```

**conrowcol**

**Purpose:** Clears all the current connections and make new row-column connections.

**Format:**

```plaintext
conrowcol(unitname, row_list, col_list)
```

- `unitname` – the matrix name in hardware configuration.
- `row_list` – list of rows to connect.
- `col_list` – list of columns to connect.

**Example: 70X**

```plaintext
conrowcol(MTRX1, 'A', 1)
conrowcol(MTRX1, {'A', 'B'}, {1, 2})
```

**Series 3700 System Switch**

```plaintext
conrowcol(MTRX1, '1', 1)
conrowcol(MTRX1, {'1', '2'}, {1, 2})
```

**crtbf**

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

**Format:**

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

`buff_name` – The name of the buffer to create.
buff_cap – The capacity of the buffer created.

append_flag – Use KI_EBAP to enable buffer append mode, and KI_DBAP to disable buffer append mode.

timestamp_flag – Use KI_EBTS to enable collecting buffer timestamps, and KI_DBTS to disable collecting buffer timestamps.

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

Format:

\[
\begin{align*}
\text{delay(second)} \\
\text{rdelay(second)}
\end{align*}
\]

delcon
Purpose: Delete terminal-pin connections.

Format:

\[
\text{delcon(*instMTRX, ter, pin, *more_pin)}
\]

instMTRX – the matrix name in hardware configuration; it's optional.

Ter – list of terminals to connect.

Pin – list of pins to connect.

more_pin – more pins to connect

Remarks: Normally, delcon () supports the row-column mode for a matrix. For more information on the how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).

Examples:

\[
\begin{align*}
\text{delcon(MTRX1,SMU1,1)} \\
\text{delcon(SMU1,1)} \\
\text{delcon(SMU1H,1)} \\
\text{delcon(SMU1L,1)} \\
\text{delcon(SMU1,1,2,3)} \\
\text{delcon({SMU1,SMU2}, \{1,2\})}
\end{align*}
\]

delconrc
Purpose: Delete connections of rows and columns in matrix.

Format:

\[
\text{delconrc(unitname, row_list, col_list)}
\]

unitname – the matrix name in hardware configuration; it's optional.

row_list – list of rows to disconnect.
col_list – list of columns to disconnect.

Remarks: row_list and col_list will be matched according to the sequence. If the numbers of row_list and col_list are not the same, connections will be disconnected according to the shorter list.

Examples: 70X

```plaintext
delconrc(MTRX1, 'A', 1)
delconrc(MTRX1, ['A', 'B'], [1, 2])
```

Series 3700 System Switch

```plaintext
delconrc(MTRX1, '1', 1)
delconrc(MTRX1, {'1', '2'}, {1, 2})
```

delpth

Purpose: Delete terminal-pin connections on specified path.

Format:

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

- `instMTRX` – the matrix name in hardware configuration; it's optional.
- `ter` – list of terminals to disconnect.
- `pin` – list of pins to disconnect.
- `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 were connected, otherwise there is no action on the Matrix.

Examples: 70X

```plaintext
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

```plaintext
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')
```

devcrl

Purpose: Sets all sources to zero.
Format:

`devclr()`

**devint**

Purpose: Resets all instruments.

Format:

`devint()`

**enable**

Purpose: Provides real-time 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 50Hz 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 to Itable[1].

**Vtable** – The table created by you; the measured value is saved to Vtable[1].
ioli/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:

```plaintext
iol(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:

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

lorangei/lorangev

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

Format:

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

measi/measv/meast

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

Format:

```plaintext
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 to Itable[1].

Ttable – The table created by you; the measured time value is saved to Ttable[1].

Vtable – The table created by you; the measured voltage value is saved to Vtable[1].

msoli/msolv/msoliv

Purpose: Measures current (msoli), voltage (msolv), or current/voltage using overlapped measurements (msoliv) using a fixed 0.001 nplc.
Format:

\begin{verbatim}
msoli(SMUX, i_buff_name)  X = SMU number (1, 2, 3, ...)
msolv(SMUX, v_buff_name)  X = SMU number (1, 2, 3, ...)
msoliv(SMUX, i_buff_name, v_buff_name)  X = SMU number (1, 2, 3, ...)
\end{verbatim}

\textbf{i_buff_name} – The buffer to store current measurements. The buffer must be created by \texttt{crtbf()}, and must be created for the same SMU.

\textbf{v_buff_name} – The buffer to store voltage measurements. The buffer must be created by \texttt{crtbf()}, and must be created for the same SMU.

\textbf{postscript}

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

Format:

\begin{verbatim}
postscript(location)
\end{verbatim}

\texttt{location} = 0: volatile memory

\texttt{location} = 1: nonvolatile memory

Default location value: 1

\textbf{postbuffer}

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

Format:

\begin{verbatim}
postbuffer("name", start_index, end_index, buff_name, avg_num)
\end{verbatim}

\texttt{name} – A string that represents the values in the script, defined by the script writer.

\texttt{start_index} – The starting index of values to post and print.

\texttt{end_index} – The ending index of values to post and print.

\texttt{buff_name} – The name of the buffer to print; it could be a default name or a user-defined name.

\texttt{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 \texttt{avg_num} value. If this parameter is not given by you, the system will give a default value of 1 (print every value point).

\textbf{postbuftime}

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

Format:

\begin{verbatim}
postbuftime("name", start_index, end_index, buff_name, avg_num)
\end{verbatim}

\texttt{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 buffer, always use the same avg_num with the one in postbuffer(), otherwise, 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 2600A.

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. The data will be posted to the data sheet in the ACS GUI.
Format:
```
posttable("name", array_name, array_start, array_end)
```

- **name** – The column heading
- **array_name** – The array that contains the data to be posted.
- **array_start** – [optional] the starting index of the array data to be posted.
- **array_end** – [optional] the ending index of the array data to be posted.

Examples:
```
posttable("col1", results) -- Return all data in results to a column in the data sheet labeled "col1"
posttable("col2", results, 3) -- Return all data in results starting from index 3 to a column in the data sheet labeled "col2"
posttable("col3", results, 3, 7) -- Return all data in results starting from index 3 and ending with index 7 to a column in the data sheet labeled "col3"
```

**rangei/rangev**

Purpose: Selects a current/voltage measurement range and prevents the selected instrument from autoranging. By selecting 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** – User-created table; the measured value is saved to Itable[1]
- **Vtable** – User-created table; the measured value is saved to 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:
\textbf{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. Also, smeasX, sintgX, or savgX must be used to set up result arrays similar to the sweep call. Each call to scnmeas will add one element to the end of each array. Calls to scnmeas may be mixed with calls to sweepX and the results will be appended to the result arrays similar to the sweepX calls.

\textbf{setauto}

Purpose: Sets SMU to autorange.

Format:

\texttt{setauto(SMUX)}  \hspace{1em} X = SMU number(1, 2, 3, ...)

\textbf{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:

\texttt{setcount(SMUX, value)}  \hspace{1em} X = SMU number(1, 2, 3, ...)

\textbf{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:

\texttt{setitv(SMUX, value)}  \hspace{1em} X = SMU number(1, 2, 3, ...)

\textbf{setmode}

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

Format:

\texttt{setmode(SMUX, modifier, value)}  \hspace{1em} X = SMU number(1, 2, 3, ...)

### Setmode parameters

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Modifier</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>smu[X]</td>
<td>KI_INTGPLC</td>
<td>&lt;value&gt; (in units of line cycles)</td>
<td>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.</td>
</tr>
<tr>
<td>KI_AVGMODE</td>
<td>KI_MEASX / KI_INTEGRATE</td>
<td></td>
<td>Controls what kind of reading is taken for avgX calls. The devint default value is KI_MEASX. When KI_INTEGRATE is specified, the time used is specified by the setmode (KI_INTGPLC) call.</td>
</tr>
<tr>
<td>KI_OFFMODE</td>
<td>KI_OFF_NORM / KI_OFF_ZERO / KI_OFF_OPEN</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>KI_MEASDELY</td>
<td>0 or KI_DELAY_OFF -1 or KI_DELAY_AUTO user value</td>
<td></td>
<td>Sets the measurement delay 0 or KI_DELAY_OFF: No delay. -1 or KI_DELAY_AUTO: Automatic delay value. user value: Set desired delay value.</td>
</tr>
</tbody>
</table>

### sintgi/sintgv

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

**Format:**

```plaintext
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 to Itable[1].

*Vtable* — The table created by you; the measured voltage value is saved to Vtable[1].

### slorangei/slorangev

**Purpose:** Defines the lowest source range that will be used during autoranging.

**Format:**

```plaintext
slorangei(SMUX, value) X = SMU number(1, 2, 3, …)
slrangev(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 table.

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 to Itable.

- **Vtable** – The table created by you; the measured voltage value is appended to Vtable.

- **Ttable** – The table created by you; the measured time value is appended to Ttable.

srangei/srangev

Purpose: Selects a current/voltage source range and prevents the selected instrument from auto-ranging. By selecting 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 in the first sweep step. 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: Initializes all SMUs (set nplc to 0.001, measure count to 1, etc.) Clear the error queues and resets all registers.

Format:
```lua
sysinit()
```

syquery

Purpose: Queries every node and every SMU in the system and gives every SMU a unique name, for instance, SMUX. It displays the node number and SMU number on every Series 2600A’s screen. Sets the integration nplc to 1 and average mode to KI_MEASX on every SMU in the system.

Format:
```lua
sysquery()
```

TSP LPT Library Command Examples

The following LPT examples are provided for your reference:

**Example 1**
```lua
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)     --set RSMU2 in 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
    elseif testmode == 1 then
        limitv(RSMU1, myLIMIT)                 --set RSMU1 voltage limit
        forcei(RSMU1, forcevalue)             --force RSMU1 current source
    end
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---

Example 2

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()
Vg_inc=(Vg_stop-Vg_start)/(Vg_points-1)
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
end   -- function
--- CALL ---
local DSMU=SMU2
local GSMU=SMU1
local SSMU=K1_GND
local BSMU=K1_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=1
local error={}       
local time={}          
local Id={}            
local Vg={}            
--- End of Input ---
Four_term_MOSFET_i_dvg(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 control one instrument or multiple instruments to perform parametric tests.

The commands in ACSLPT can be used to control some general instruments. To use these commands, first, you need to import ACSLPT to a PTM. The commands can control the following instruments: Series 23x, Series 2400 SourceMeter instruments, Series 2600A SourceMeter instruments, Series 3700 System Switch, Model 4200 CVU, Model 4200/4210 SMU, Switch Matrix 707/707A/707B and 708/708A/708B, and LCR 4284/4980 capacitance meter. For more information about Configuring a Python Language Test Module (PTM), refer to the ACS Reference Manual (document number: ACS-901-01).

![Figure 18: ACS LPT call flow](image)

**NOTE**

In the following table, you will learn how the CTM modules and the ACS software function and interact.
ACS software compatibility

<table>
<thead>
<tr>
<th>ACS installed on:</th>
<th>Interface:</th>
<th>Compatible library:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model 4200-SCS</td>
<td>Normal (non-KXCI)</td>
<td>CTM functions</td>
</tr>
<tr>
<td>Model 4200-SCS</td>
<td>KXCI and Ethernet cable</td>
<td>Ki42cvulpt commands</td>
</tr>
<tr>
<td>PC</td>
<td>KXCI and Ethernet cable</td>
<td>Ki42cvulpt commands</td>
</tr>
</tbody>
</table>

Python LPT functions

In the following tables, function calls are grouped according to instrument type. The details on functions for the SMUs and general operations are listed alphabetically.

LPT functions

**Models 236, 237, 238 LPT functions**

- devclr
- devint
- forcei
- forcev
- intgi
- intgv
- limiti
- limitv
- lorangei
- lorangev
- measi
- measv
- rangei
- rangev
- setauto
- setmode
- srangei
- srangev

**NOTE**

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

LPT functions

**Series 2400 SourceMeter instruments LPT functions**

- abort
- bsweepi
- bsweepv
- delay
- devclr
- devint
- forcei
- forcev
- intgi
- intgv
- limiti
- limitv
- measi
- measv
- rangei
- rangev
- setauto
- setmode
- srangei
- srangev
- sweepi
- sweepv
- trigig
- trigil
- trigtg
- trigtl
- trigvg
- trigvl
### LPT functions

#### Series 2600A SourceMeter instruments LPT functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>abort</td>
<td></td>
</tr>
<tr>
<td>devclr</td>
<td>devint</td>
</tr>
<tr>
<td>intgv</td>
<td>limiti</td>
</tr>
<tr>
<td>measi</td>
<td>measv</td>
</tr>
<tr>
<td>setmode</td>
<td>srangei</td>
</tr>
<tr>
<td>trigtg</td>
<td>trigtl</td>
</tr>
</tbody>
</table>

#### Series 3700 System Switch LPT functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addcon</td>
<td>addconrc</td>
</tr>
<tr>
<td>conpin</td>
<td>conpth</td>
</tr>
<tr>
<td>delcon</td>
<td>delpth</td>
</tr>
</tbody>
</table>

#### Model 4200-SCS LPT functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>avgv</td>
<td>avgv</td>
</tr>
<tr>
<td>delay</td>
<td>devclr</td>
</tr>
<tr>
<td>enable</td>
<td>execut</td>
</tr>
<tr>
<td>intgi</td>
<td>intgv</td>
</tr>
<tr>
<td>lorangei</td>
<td>lorangev</td>
</tr>
<tr>
<td>measz</td>
<td>rangei</td>
</tr>
<tr>
<td>setauto</td>
<td>setfreq</td>
</tr>
</tbody>
</table>

#### Models 707, 708 LPT functions

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addcon</td>
<td>addconrc</td>
</tr>
<tr>
<td>conpin</td>
<td>conpth</td>
</tr>
</tbody>
</table>

---
### LPT functions

#### Models 707, 708 LPT functions
- delconrc
- delpth
- devint

#### Model 4200 CVU LPT functions
- devclr
- devint
- forcev
- measz
- rangei
- setauto
- setfreq
- setlevel
- setmode

#### 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 ptmlpt.constantlp to the header lines of a PTM (see next Figure). The ACSLPT commands are listed in alphabetical order.

**Figure 19: Import ACSLPT**

```
from ACS_PostData import *
from ACSLPT import *
from ptmlpt.constantlp import *
```

**abort**

Purpose: To abort the current source-delay-measure process. Recommend you use or call abort function node in UAP only.
NOTE

The abort function is only valid with the Series 2600A (KI2600) instruments and Series 2400 (KI2400) 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 – list of terminals to connect.

pin – list of pins to connect.

more_pin – 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 shorter list.

Normally, addcon supports the row-column mode of matrix. When the matrix is set to instrument-card mode, a row is assigned automatically to connect the terminal and pin.

For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).

Examples:

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

addconrc

Purpose: Only add connection of rows and columns in matrix.

Format:

addconrc(unitname, row_list, col_list)

unitname – the matrix name in hardware configuration.
row_list – list of rows to connect.

col_list – list of columns to connect.

Remarks: For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).

Examples: 70X

\[
\text{addconrc(MTRX1, 'A', 1)} \\
\text{addconrc(MTRX1, ['A', 'B'], [1, 2])}
\]

Series 3700 System Switch

\[
\text{addconrc(MTRX1, '1', 1)} \\
\text{addconrc(MTRX1, ['1', '2'], [1, 2])}
\]

\text{addpth}

Purpose: Add terminal-pin connections by path.

Format:

\[
\text{addpth(*instMTRX, ter, pin, row)}
\]

\text{instMTRX} – the matrix name in hardware configuration; it's optional.

\text{Ter} – list of terminals to connect.

\text{Pin} – list of pins to connect.

\text{Row} – the row used to connect terminals and pins.

Examples: 70X

\[
\text{addpth(MTRX1, SMU1, 1, 'A')} \\
\text{addpth(SMU1, 1, 'A')} \\
\text{addpth(SMU1H, 1, 'A')} \\
\text{addpth(SMU1L, 1, 'A')} \\
\text{addpth([SMU1, SMU2], [1, 2], 'A')} \\
\text{addpth(MTRX1, [SMU1, SMU2], [], 'A')} \\
\text{addpth([], [1, 2], 'A')}
\]

Series 3700 System Switch

\[
\text{addpth(MTRX1, SMU1, 1, '1')} \\
\text{addpth(SMU1, 1, '1')} \\
\text{addpth(SMU1H, 1, '1')} \\
\text{addpth(SMU1L, 1, '1')} \\
\text{addpth([SMU1, SMU2], [1, 2], '1')} \\
\text{addpth(MTRX1, [SMU1, SMU2], [], '1')} \\
\text{addpth([], [1, 2], '1')}
\]

Remarks: All terminals and pins will be connected together in the row. The command cannot connect paths in multiple matrices.
In addition, you can only connect terminals or pins with this function. However, when only connecting terminals, the instMTRX is required, otherwise, the function will not know which command to send to the instrument.

For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).

**Figure 20: addpth library command**

![Diagram of addpth function](image)

---

### `avgi/avgv`

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

**Format:**

```plaintext
avgi(unit name, stepNo, dStepTime)
avgv(unit name, stepNo, dStepTime)
```

- `stepNo` – The number of steps averaged in the measurement. This number ranges from 1 to 160,000 (Model 4200 is 32767).
- `dStepTime` – The interval in seconds between each measurement. Minimum practical time is approximately 0.0001s (nplc must be set to 0.001, and Model 4200 set to 2.5us).

**Example:**

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

### `bsweepx`

**Purpose:** Supplies a series of ascending or descending voltages or currents and shuts down the source when a trigger condition is encountered.

**Format:**

```plaintext
bsweep(SMUX, startval, endval, num_points, delay_time, result);
bsweepv(SMUX, startval, endval, num_points, delay_time, result);
```

- `startval` – The initial voltage or current level applied as the first step in the sweep. This value can be positive or negative.
- `endval` – The final voltage or current level applied as the last step in the sweep. This value can be positive or negative.
num_points – The number of separate current and voltage force points between startval and endval.

delay_time – The delay in seconds between each step and the measurements defined by the active measure list.

Result – Assigned to the result of the trigger. This value represents the source value applied at the time of the trigger or breakdown.

checkparam

Purpose: Check the hardware limits parameter according to hwlimits file. The command is only used for 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. It is used to pass a keyword, variable-length argument list. The names are the same as 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:
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 dictionary.

Example:
checkparam(SMU1, dc_lmti_i = 1, dc_srange_v = 10)

clrattrset

Purpose: Clear current instrument setting saved in memory.

Format:
clrattrset(*args)

*args – a tuple of arbitrary positional arguments. It is used to pass a non-keyword, variable-length argument list.

Example:
clrattrset(SMU1, SMU2)
**clrcon**

Purpose: Clear all connections matrix.

Format:

```c
clrcon(unitname)
```

*unitname* – The instrument name in \ACS\KATS\CONFIG\ACS hdcon Online.kcf. For example, MTRX1.

Example:

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

**Figure 21: clrcon library command**

![Diagram of clrcon](image)

**clrscn**

Purpose: Clears previously defined sweep measurement.

Format:

```c
clrscn(*args)
```

*args – a tuple of arbitrary positional arguments. It is used to pass a non-keyword, variable-length argument list.

Example:

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

**clrtrg**

Purpose: Clears previously defined sweep trigger setting. This permits the use of trigXl or trigXg more than once with different levels within a single test sequence.

Format:

```c
clrtrg(*args)
```
Figure 22: clrtrg library command

Example:

```plaintext
conpin(SMU1, 1)
addcon(GND, 2)
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.
clrscl() # Clear sweepv
trigil(SMU1, -0.5e-3)# Decrease ramp to I = -0.5mA.
smeasi(SMU1, forcur)# Measure reverse
sweepv(SMU1, 0.0, -30.0, 10, 5.00e-3)
```

conpin

Purpose: Clears all the current connections and makes new terminal-pin connections.

Format:

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

`instMTRX` – the matrix name in hardware configuration; it’s optional.

`Ter` – list of terminals to connect.

`Pin` – list of pins to connect.

`more_pin` – more pins to connect

Remarks: Normally, `conpin()` supports row-column mode of matrix. When matrix is set to instrument-card mode, rows will be assigned automatically to connect the terminals and pins.

For more information on how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).
Example:

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

**conpth**

Purpose: Clears all the current connections and makes new terminal-pin connections by path.

Format:

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

*inst MTRX — the matrix name in hardware configuration; it’s optional.

**Ter** — list of terminals to connect.

**Pin** — list of pins to 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 matrices.

Also, you can only connect terminals or pins with this function. However, when only connecting terminals, the instMTRX is required, otherwise, the function will not know which command to send to the instrument.

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')
```
**conrowcol**

Purpose: Clears all the current connections and makes new row-column connections.

Format:

```
conrowcol(unitname, row_list, col_list)
```

- **unitname** – the matrix name in hardware configuration.
- **row_list** – list of rows to connect.
- **col_list** – list of columns to connect.

Examples: 70X

```
conrowcol(MTRX1, 'A', 1)
conrowcol(MTRX1, ['A', 'B'], [1, 2])
```

**Series 3700 System Switch**

```
conrowcol(MTRX1, '1', 1)
conrowcol(MTRX1, ['1', '2'], [1, 2])
```

**delay**

Purpose: Provides user-programmable delay within a test sequence. The unit is 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** – list of terminals to connect.
- **Pin** – list of pins to connect.
- **more_pin** – more pins to connect

Remarks: Normally, delcon() supports the row-column mode for a matrix. For more information on the how to set the instrument-card mode and row-column mode, refer to the ACS Reference manual (document number: ACS-901-01).

Examples:

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

delconrc

Purpose: Delete connections of rows and columns in matrix.

Format:
```
delconrc(unitname, row_list, col_list)
```

- **unitname** – the matrix name in hardware configuration; it's optional.
- **row_list** – list of rows to disconnect.
- **col_list** – list of columns to disconnect.

Remarks: row_list and col_list will be matched according to the sequence. If the numbers of row_list and col_list are not the same, connections will be disconnected according to the shorter list.

Examples: 70X
```
delconrc(MTRX1, 'A', 1)
delconrc(MTRX1, ['A','B'],[1,2])
```

Series 3700 System Switch
```
delconrc(MTRX1, '1',1)
delconrc(MTRX1, ['1','2'],[1,2])
```

delpth

Purpose: Delete terminal-pin connections on specified path.

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

- **instMTRX** – the matrix name in hardware configuration; it's optional.
- **Ter** – list of terminals to disconnect.
- **Pin** – list of pins to disconnect.
- **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 were connected, otherwise there is no action on the Matrix.

Examples: 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


```plaintext
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 zero state.

**Format:**

```plaintext
devclr(*args)
```

**Examples:**

```plaintext
devclr()    ## Clear all instruments
devclr(SMU1)   ## Clear SMU1
devclr(SMU1, CVU1) ## Clear SMU1 and the CVU
```

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 supported instruments, GPIB based instruments will be cleared by sending all strings defined as kibdefclr. The devclr is implicitly called by clrcon, devint, execut, and tstdsl.

**devint**

Purpose: Reset the instruments and clear the matrix by opening all relays and disconnecting the pathways. Meters and sources are reset to the default states.

**Format:**

```plaintext
devint(*args)
```

**Examples:**

```plaintext
devint()    ## Reset all instruments
devint(SMU1)   ## Reset SMU1
devint(SMU1, SMU3) ## Reset SMU1  and SMU3
```

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 devint function will execute as follows:

- Clear all sources.
- Stop outputting pulses from the pulse generator cards.
- Reset all instruments in the system to their default states.

For non-4200-SCS instruments, the following actions will be performed prior to resetting the instruments:

- Clear all sources by calling devclr.
- Clear the matrix cross-points by calling clrcon.
Clear the trigger tables by calling clrtrg.
Clear the sweep tables by calling clrsrn.
Reset GPIB instruments by sending the string defined with kibdefint (if used).

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

**NOTE**

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

<table>
<thead>
<tr>
<th>Default settings</th>
<th>Series 2600A SourceMeter instruments setting after devint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>turn source off</td>
</tr>
<tr>
<td>Reset</td>
<td>reset all bits of following register to 0:</td>
</tr>
<tr>
<td></td>
<td>• standard event register</td>
</tr>
<tr>
<td></td>
<td>• operation event register</td>
</tr>
<tr>
<td></td>
<td>• measurement event register</td>
</tr>
<tr>
<td></td>
<td>• questionable event register</td>
</tr>
<tr>
<td>Long-form and short-form versions</td>
<td>command word will be sent in short-form version</td>
</tr>
<tr>
<td>ACS hardware configuration setting window</td>
<td>if interlock enabled enable interlock</td>
</tr>
<tr>
<td></td>
<td>if rear panel enabled enable rear panel</td>
</tr>
<tr>
<td></td>
<td>if beeper disabled disable beeper</td>
</tr>
</tbody>
</table>
**Default settings**

**Series 2400 SourceMeter instruments setting after devint**

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: Stop the timer and set the time value to zero. Timer reading is also stopped.

Format:
```
disable(unitname)
```

unitname – the instrument ID of timer module (TIMERn)

Example:
```
disable('TIMER1')
```

**enable**

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

Format:
```
execut(*args)
```
Examples:

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

Remarks: For the Model 4200 or Series 2600A instruments, this function will wait for all previous LPT commands to complete and then will perform a devint.

**forcei/forcev**

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

Format:

```plaintext
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 amps.

**getcommon**

Purpose: Get the common attribute of the current hardware from the global_dict. Return key list: [UNITLIST, PLC, pin]

Format:

```plaintext
getcommon()
```

Example:

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

**getinstattr**

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

Format:

```plaintext
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 an instrument name.
Return Values:

- `INVAL_INST_ID` - the instrument specified is not in the configuration
- `-1` - this function does not apply to this unit
- `None` - The attribute specified does not exist for the selected instrument.
- `attribute value` - The returned attribute

Example:

```python
address = getinstattr(SMU1, "GPIB_ADDRESS") ## Returns the GPIB Address of SMU1
print getinstattr(SMU1, "MODEL")    ## Prints the Model of SMU1 to the log window
```

**getinstid**

Purpose: Get the instrument identifier (ID) from the instrument name string. This function is only supported when ACS is installed on the 4200-SCS and can only be used with Model 4200 instruments.
Format:

\texttt{getinstid(unitname)}

\texttt{unitname} – The instrument name in $\backslash$ACS\KATS\CONFIG\ACS_hdcon_Online.kcf.

Return Value:

instrument identifier (ID)

Example:

\texttt{print getinstid(SMU1)}

\texttt{4100}

\textbf{getstatus}

Purpose: Returns the operating state of the specified instrument.

Format:

\texttt{getstatus(unitname, iCode)}

\texttt{unitname\_str} – The instrument name in $\backslash$ACS\KATS\CONFIG\ACS_hdcon_Online.kcf.

\texttt{iCode} – The parameter to 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 an SMU is provided in the next table.

<table>
<thead>
<tr>
<th>iCode</th>
<th>Comment</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_IPVALUE</td>
<td>The presently programmed output value</td>
<td>Current value (I output value)</td>
</tr>
<tr>
<td>KI_VPVALUE</td>
<td>The presently programmed output value</td>
<td>Voltage value (V output value)</td>
</tr>
<tr>
<td>KI_IPRANGE</td>
<td>The presently programmed range</td>
<td>Current range (full-scale range value, or 0.0 for autorange)</td>
</tr>
<tr>
<td>KI_VPRANGE</td>
<td>The presently programmed range</td>
<td>Voltage range (full-scale range value, or 0.0 for autorange)</td>
</tr>
<tr>
<td>KI_IARANGE</td>
<td>The presently active range</td>
<td>Current range (full-scale range value)</td>
</tr>
</tbody>
</table>
## Getstatus parameters

<table>
<thead>
<tr>
<th>iCode</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>KI_VARANGE</code></td>
<td>Voltage range (full-scale range value)</td>
</tr>
<tr>
<td><code>KI_IMRANGE</code></td>
<td>For autorange, the range at which the previous I measurement was performed.</td>
</tr>
<tr>
<td><code>KI_VMRANGE</code></td>
<td>For autorange, the range at which the previous V measurement was performed.</td>
</tr>
<tr>
<td><code>KI_COMPLNC</code></td>
<td>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).</td>
</tr>
<tr>
<td><code>KI_RANGE_COMPLIANCE</code></td>
<td>Active compliance status for fixed range In range compliance if 1</td>
</tr>
</tbody>
</table>

### 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 devices.

#### Example:

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

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

Format:

```python
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. The max_size can be a number from 0 to 65535 (hex FFFF).

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

#### Example:

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

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

Format:

```python
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 that 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 success or failure of the command.

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:
```
poll = 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: Perform voltage or current measurement 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 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:
\begin{verbatim}
    i1 = intgi(SMU1)
\end{verbatim}

limiti/limitv

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

Format:
\begin{verbatim}
    limiti(unitname,dValue)
    limitv(unitname,dValue)
\end{verbatim}

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 10.0V. A limiti ("SMU1", 1.5E-3) limits the current of the voltage source of SMU1 to 1.5mA.

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 lowest autorange limit.

Format:
\begin{verbatim}
    lorangei(unitname,dValue)
    lorangev(unitname,dValue)
\end{verbatim}

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

dValue – The value of the desired instrument range, in volts or amps.

Remarks: lorange is used to set the lowest autorange range. Autorange will start from the setup value. This function can save test time.

For the Model 4200, if the instrument is 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 will take effect until a devclr, devint, execut, or another lrangeX executes.

For the Series 23x instruments, the function works as autorange. The second dValue will be ignored.

It cannot be used for the Series 2400 SourceMeter instruments.

Example:
\begin{verbatim}
    lorangei(SMU1, 2.0E-6)
\end{verbatim}
**measi/measv**

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

**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 = 166.67µs (0.01PLC/60Hz). For 50Hz line power, 0.01PLC = 200µs (0.01PLC/50Hz).

**Example:**

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

**measz**

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

**Format:**

```plaintext
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 Table).

- **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

<table>
<thead>
<tr>
<th>unitname_str (Model name)</th>
<th>iModel (Measurement Model)</th>
<th>parameter values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CVU1</td>
<td>ZTH Impedance (Z) and phase ((\phi) in radians)</td>
<td>KI_CVU_TYPE_ZTH or 0</td>
</tr>
<tr>
<td></td>
<td>RJX Resistance and reactance</td>
<td>KI_CVU_TYPE_RJX or 1</td>
</tr>
<tr>
<td></td>
<td>CpGp Parallel capacitance and conductance</td>
<td>KI_CVU_TYPE_CPGP or 2</td>
</tr>
<tr>
<td></td>
<td>CsRs Series capacitance and resistance</td>
<td>KI_CVU_TYPE_CSRS or 3</td>
</tr>
<tr>
<td></td>
<td>CpD Parallel capacitance and dissipation factor</td>
<td>KI_CVU_TYPE_CPD or 4</td>
</tr>
<tr>
<td></td>
<td>CsD Series capacitance and dissipation factor</td>
<td>KI_CVU_TYPE_CSD or 5</td>
</tr>
<tr>
<td></td>
<td>RAW Raw data from measure</td>
<td>KI_CVU_TYPE_RAW or 6</td>
</tr>
<tr>
<td>CMTR1</td>
<td>Z-thr Impedance (Z) and phase ((\phi) in radians)</td>
<td>KI_AGCV_TYPE_CPD or 0</td>
</tr>
<tr>
<td></td>
<td>R-X Resistance and Reactance</td>
<td>KI_AGCV_TYPE_RX or 1</td>
</tr>
<tr>
<td></td>
<td>Cp-G Parallel capacitance and equivalent parallel conductance</td>
<td>KI_AGCV_TYPE_CPG</td>
</tr>
<tr>
<td></td>
<td>Cs-Rs Series capacitance and resistance</td>
<td>KI_AGCV_TYPE_CSRS</td>
</tr>
<tr>
<td></td>
<td>Cp-D Parallel capacitance and dissipation factor</td>
<td>KI_AGCV_TYPE_CPD</td>
</tr>
<tr>
<td></td>
<td>Cs-D Series capacitance and dissipation factor</td>
<td>KI_AGCV_TYPE_CSD</td>
</tr>
<tr>
<td></td>
<td>Cp-Q Parallel capacitance and Quality factor (inverse of D)</td>
<td>KI_AGCV_TYPE_CPD</td>
</tr>
<tr>
<td></td>
<td>Cs-Q Series capacitance and Quality factor (inverse of D)</td>
<td>KI_AGCV_TYPE_CSQ</td>
</tr>
<tr>
<td></td>
<td>Lp-D Inductance value measured with parallel-equivalent circuit Model and dissipation factor</td>
<td>KI_AGCV_TYPE_LPD</td>
</tr>
<tr>
<td></td>
<td>Lp-Q Inductance value measured with parallel-equivalent circuit Model and Quality factor (inverse of D)</td>
<td>KI_AGCV_TYPE_LPQ</td>
</tr>
<tr>
<td></td>
<td>Lp-G Parallel inductance value and equivalent parallel conductance</td>
<td>KI_AGCV_TYPE_LPG</td>
</tr>
<tr>
<td></td>
<td>Lp-Rp Parallel inductance value and Equivalent parallel resistance</td>
<td>KI_AGCV_TYPE_LPRP</td>
</tr>
<tr>
<td></td>
<td>Ls-D Series inductance value and dissipation factor</td>
<td>KI_AGCV_TYPE_LSD</td>
</tr>
<tr>
<td></td>
<td>Ls-Q Series inductance value and Quality factor (inverse of D)</td>
<td>KI_AGCV_TYPE_LSQ</td>
</tr>
<tr>
<td></td>
<td>Ls-Rs Series inductance value and equivalent resistance</td>
<td>KI_AGCV_TYPE_LSRS</td>
</tr>
<tr>
<td></td>
<td>Z-thd Impedance (Z) and phase ((\phi) in degree)</td>
<td>KI_AGCV_TYPE_ZTD</td>
</tr>
<tr>
<td></td>
<td>Cp-Rp Parallel capacitance and equivalent resistance</td>
<td>KI_AGCV_TYPE_CPRP</td>
</tr>
<tr>
<td></td>
<td>G-B Equivalent parallel conductance and Susceptance</td>
<td>KI_AGCV_TYPE_GB</td>
</tr>
<tr>
<td></td>
<td>Y-thd Admittance and phase (in degree)</td>
<td>KI_AGCV_TYPE_YTD</td>
</tr>
<tr>
<td></td>
<td>Y-thr Admittance and phase (in radians)</td>
<td>KI_AGCV_TYPE_YTR</td>
</tr>
</tbody>
</table>
Measurement settings

<table>
<thead>
<tr>
<th>unitname_str (Model name)</th>
<th>iModel (Measurement Model)</th>
<th>parameter values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vdc-Idc</td>
<td>Direct-current voltage and Direct-current electricity</td>
<td>KI_AGCV_TYPE_VDID</td>
</tr>
</tbody>
</table>

Example:

```python
testData = measz(CVU1, KI_CVU_TYPE_CSRS, KI_CVU_SPEED_NORMAL)
```

rangei/rangev

**Purpose:** Selects measurement range and prevents the selected instrument from autoranging. When you select a range, the time required for autoranging is eliminated.

**Format:**

```python
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 for the Series 2600A SourceMeter instruments.

Example:

```python
rangei(SMU1, 2.0E-3)  # Set current range to 2mA.
```

rdelay

**Purpose:** A user-programmable delay in seconds.

**Format:**

```python
rdelay(dDelayTime)
```

- `dDelayTime` — The desired delay duration in seconds.

Example:

```python
rdelay(0.02)# Pause for 0.02s
```

setauto

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

**Format:**

```python
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 (voltage versus current) setauto places both voltage and current ranges in autorange mode.

Example:

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

**setfreq**

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

Format:

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

- `unitname` – The instrument name in `\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`. Only “CVUn” and “CMTRn” are supported.

- `dFreq` – Frequency of the AC drive in Hz.

Example:

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

**setlevel**

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

Format:

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

- `unitname` – The instrument name in `\ACS\KATS\CONFIG\ACS_hdcon_Online.kcf`. Only “CVUn” and “CMTRn” are supported.

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

Example:

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

**setmode**

Purpose: Set instrument specific operating mode parameters.

Format:

```plaintext
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.
<table>
<thead>
<tr>
<th>Setmode</th>
<th>Series 23x LPT Parameters</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>unitname</strong></td>
<td><strong>iModifier</strong></td>
<td><strong>dValue</strong></td>
</tr>
<tr>
<td><em>str</em></td>
<td>KI_INTGPLC</td>
<td><em>value</em> (in units of line cycles)</td>
</tr>
<tr>
<td>KI_SENSE</td>
<td>KI_SENSE_LOCA (or 0)</td>
<td>KI_SENSE_REMO (or 1)</td>
</tr>
<tr>
<td>KI_TRIG_IN</td>
<td>KI_TRIG_IN_CONT = 0</td>
<td>KI_TRIG_IN_SRC = 1</td>
</tr>
<tr>
<td></td>
<td>Input triggers. Input trigger are used to control when source, delay, and measure operations occur:</td>
<td></td>
</tr>
<tr>
<td>KI_TRIG_IN_CONT: Continuously process all SDM(Source Delay Measure) cycles.</td>
<td>KI_TRIG_IN_SRC: Each trigger will process an SDM cycle.</td>
<td>KI_TRIG_IN_DLY: Initial trigger sets source. Each subsequent trigger initiates a delay and measure then sets source of next SDM cycle.</td>
</tr>
</tbody>
</table>
### Setmode

#### Series 23x LPT Parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_TRIG_SOURCE</td>
<td>KI_TRIG_X</td>
<td>0 Input trigger origin. The input trigger stimulus may be provided by front</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_GET</td>
<td>1 manual trigger function, and external device that applies a TTL level pulse to</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_TALK</td>
<td>2 the TRIGGER in connector on the rear panel, or an appropriate IEEE-488</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_EXTERNAL</td>
<td>3 operation.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_INTERNAL</td>
<td>4</td>
</tr>
<tr>
<td>KI_TRIG_OUT</td>
<td>KI_TRIG_OUT_NONE</td>
<td>0 Output trigger generation:</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_SRC</td>
<td>1 KI_TRIG_OUT_NONE: No output triggers.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_DLY</td>
<td>2 KI_TRIG_OUT_SRC: Output trigger pulse after every source phase.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_SRCDLY</td>
<td>3 KI_TRIG_OUT_DLY: Out put trigger pulse after every delay phase.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_MSR</td>
<td>4 KI_TRIG_OUT_SRCDLY: Out put trigger pulse after every source phase and delay</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_SRCMSR</td>
<td>5 phase.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_DLYMSR</td>
<td>6 KI_TRIG_OUT_MSR: Out put trigger pulse after every source phase and measure</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_SRCDLYMSR</td>
<td>7 phase.</td>
</tr>
<tr>
<td></td>
<td>KI_TRIG_OUT_PULSE</td>
<td>8 KI_TRIG_OUT_SRCMSR: Output trigger pulse after every source phase and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>measure phase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KI_TRIG_OUT_SRCDLYMSR: Output trigger pulse after every source phase, delay</td>
</tr>
<tr>
<td></td>
<td></td>
<td>phase and measure phase.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>KI_TRIG_OUT_PULSE: For pulse sweeps . Output trigger pulse after end of each</td>
</tr>
<tr>
<td></td>
<td></td>
<td>off time measure.</td>
</tr>
<tr>
<td>KI_SWEEPEND_TRIGO</td>
<td>KI_SWEEPEND_TRIGOUT_EN</td>
<td>1 When enable, an output trigger pulse occurs at the end of the sweep.</td>
</tr>
<tr>
<td>UT</td>
<td>KI_SWEEPEND_TRIGOUT_DIS</td>
<td>0</td>
</tr>
<tr>
<td>KI_AVGNUMBER</td>
<td>0, 2, 4, 8, 16, 32</td>
<td>Number of readings to take average. 0 means disable average filter</td>
</tr>
</tbody>
</table>
## Setmode

### Series 2400 instruments LPT parameters

<table>
<thead>
<tr>
<th>Unitname_str</th>
<th>iModifier</th>
<th>dValue</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMU1</td>
<td>KI_INTGPLC</td>
<td>&lt;value&gt; (in units of line cycles)</td>
<td>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<del>10(DC) and 0.004</del>0.1(2430 Pulse mode).</td>
</tr>
<tr>
<td>SMU1 (only 2430 SMU)</td>
<td>KI_TRIG_IN_CONT</td>
<td>&lt;value&gt;</td>
<td>Set the output pulse count.</td>
</tr>
<tr>
<td>SMU1</td>
<td>PULSE_MODE_PULSE</td>
<td>VOLT CURR</td>
<td>Click pulse mode and pulse source function VOLT: voltage source VOLT: voltage source CURR: current source</td>
</tr>
<tr>
<td></td>
<td>PULSE_MODE_WID</td>
<td>&lt;value&gt;</td>
<td>Click pulse mode and set pulse width</td>
</tr>
<tr>
<td></td>
<td>PULSE_MODE_DELA</td>
<td>&lt;value&gt;</td>
<td>Click pulse mode and set pulse delay</td>
</tr>
</tbody>
</table>

### Series 2600A instruments LPT parameters

<table>
<thead>
<tr>
<th>unitname_str</th>
<th>iModifier</th>
<th>dValue</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMU1</td>
<td>KI_INTGPLC</td>
<td>&lt;value&gt; (in units of line cycles)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>KI_AVGMODE</td>
<td>KI_MEASX KI_INTEGRATE</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>KI_OFFMODE</td>
<td>KI_OFF_NORM KI_OFF_ZERO KI_OFF_OPEN</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Setmode

**Series 2600A instruments LPT parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value(s)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_MEASDELY</td>
<td>0 or KI_DELAY_OFF - 1 or KI_DELAY_AUTO</td>
<td>No delay. Automatic delay value. Set user delay value.</td>
</tr>
<tr>
<td></td>
<td>user value</td>
<td></td>
</tr>
</tbody>
</table>

**Setmode**

**Model 4200 CVU LPT parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value(s)</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>unitname_str</td>
<td>iModifier</td>
<td>dValue</td>
</tr>
<tr>
<td>KI_CVU_COMPENSATE</td>
<td>[1,1,1]</td>
<td>Set open_comp, short_comp, load_comp in one command. Value must be list type and have three items.</td>
</tr>
<tr>
<td></td>
<td>To each item:</td>
<td>0=OFF</td>
</tr>
<tr>
<td></td>
<td>1=ON</td>
<td></td>
</tr>
<tr>
<td>CVU1</td>
<td>KI_CVU_CABLE_CORRECT</td>
<td>0, 1.5 or 3</td>
</tr>
<tr>
<td></td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>KI_CVU_CUST_SPEED</td>
<td>[delay_factro, filter_factor, aperture]</td>
<td>Select customize speed mode and set parameter value, including delay_factor, filter_factor, and aperture. Value must be list type and have three items.</td>
</tr>
<tr>
<td></td>
<td>delay_factor: 0<del>100, aperture: 0006</del>10.002</td>
<td></td>
</tr>
<tr>
<td>KI_CVU_OPEN_COMPENSATE</td>
<td>0=OFF</td>
<td>Enable or disable compensation constants for open load and short.</td>
</tr>
<tr>
<td>KI_CVU_SHORT_COMPENSATE</td>
<td>1=ON</td>
<td></td>
</tr>
<tr>
<td>KI_CVU_LOAD_COMPENSATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KI_CVU_FILTER_FACTOR</td>
<td>0 to 100</td>
<td>Set the custom speed filter factor.</td>
</tr>
<tr>
<td>KI_CVU_MEASURE_SPEED</td>
<td>KI_CVU_SPEED_FAST=0 KI_CVU_SPEED_NORMA</td>
<td>Set CVU speed</td>
</tr>
<tr>
<td></td>
<td>L=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KI_CVU_SPEED_QUIET=2 KI_CVU_SPEED_CUSTO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>M=3</td>
<td></td>
</tr>
<tr>
<td>KI_CVU_MEASURE_MODEL</td>
<td>KI_CVU_TYPE_ZTH=0 KI_CVU_TYPE_RJX=1</td>
<td>For more information on the CVU mode type (see measz).</td>
</tr>
<tr>
<td></td>
<td>KI_CVU_TYPE_CPGP=2 KI_CVU_TYPE_CSRS=3</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KI_CVU_TYPE_CPD=4 KI_CVU_TYPE_CSD=5</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KI_CVU_TYPE_RAW=6</td>
<td></td>
</tr>
<tr>
<td>KI_CVU_CHANNEL</td>
<td>1~8</td>
<td>Selects CVU card on which subsequent card CVU commands will act.</td>
</tr>
<tr>
<td>KI_CVU_OFFSET</td>
<td>-30~30</td>
<td>Apply offset value to the DC low terminal.</td>
</tr>
</tbody>
</table>
### Setmode

**Model 4200 CVU LPT parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value 1</th>
<th>Value 2</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_CVU_ACVHI</td>
<td>1</td>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>KI_CVU_DCVHI</td>
<td>1</td>
<td>2</td>
<td>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.</td>
</tr>
</tbody>
</table>
| KI_CVU_MODE           | 0, 1    |         | 0: set CVU to user mode
|                       |         |         | 1: set CVU to system mode                                                    |

### Setmode

**Model 4284 LPT parameters**

<table>
<thead>
<tr>
<th>Unitname</th>
<th>Modifier</th>
<th>Value</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMTR1</td>
<td>KI_CVU_CABLE_CORRECT</td>
<td>0, 1.5  or 3</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>KI_CVU_OPEN_COMPENSATE</td>
<td>0=OFF</td>
<td>Enable or disable compensation constants for open load and short.</td>
</tr>
<tr>
<td></td>
<td>KI_CVU_SHORT_COMPENSATE</td>
<td>1=ON</td>
<td></td>
</tr>
<tr>
<td></td>
<td>KI_CVU_LOAD_COMPENSATE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>KI_CVU_FILTER_FACTOR</td>
<td>0 to 100</td>
<td>Set the custom speed filter factor</td>
</tr>
<tr>
<td></td>
<td>KI_CVU_MEASURE_SPEED</td>
<td></td>
<td>Set CVU speed</td>
</tr>
<tr>
<td></td>
<td>KI_CVU_MEASURE_MODEL</td>
<td></td>
<td>For more information on the CVU mode type see measz.</td>
</tr>
</tbody>
</table>
|            | KI_CVU_MODE               | 0 or 1  | 0: set CVU to user mode
|            |                           |         | 1: set CVU to system mode                                               |
### Setmode

Model 4284 LPT parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_AGCV_CORRECT_METHOD</td>
<td>HOD_MULT = 0</td>
<td>Elects the correction mode (Single or Multi). Scanner I/F should be installed for the Multi mode</td>
</tr>
<tr>
<td></td>
<td>HOD_SING = 1</td>
<td></td>
</tr>
<tr>
<td>KI_AGCV_CORRECT_METHOD_SING</td>
<td></td>
<td>Sets the correction mode to “SINGLE”</td>
</tr>
<tr>
<td>KI_AGCV_CORRECT_METHOD_MLT</td>
<td></td>
<td>Sets the correction mode to “MULTI”</td>
</tr>
<tr>
<td>KI_AGCV_TRIG_SOURCE</td>
<td>KI_AGCV_TRIG_I NTERNAL = 0</td>
<td>Selects the trigger mode: KI_AGCV_TRIG_INTERNAL: Sets trigger source to “internal”</td>
</tr>
<tr>
<td></td>
<td>KI_AGCV_TRIG_HOLD = 1</td>
<td>KI_AGCV_TRIG_HOLD: Sets trigger source to “manual”</td>
</tr>
<tr>
<td></td>
<td>KI_AGCV_TRIG_EXTERNAL = 2</td>
<td>KI_AGCV_TRIG_EXTERNAL: Sets trigger source to “external connector on the rear panel”</td>
</tr>
<tr>
<td></td>
<td>KI_AGCV_TRIG_BUS = 3</td>
<td>KI_AGCV_TRIG_BUS: Sets trigger source to “GPIB/LAN/USB”</td>
</tr>
<tr>
<td>KI_AGCV_INIT_CONTINUE</td>
<td>0=OFF 1=ON</td>
<td>Enables the automatic trigger to change state from the “Idle” state to the “Wait for Trigger” state. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ON or 1: Enables automatic trigger state change OFF or 0 (Preset value): Disables automatic trigger state change</td>
</tr>
</tbody>
</table>
## Setmode

### Model 4284 LPT parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_AGCV_DISPLAY_PAGE</td>
<td>0</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_BNUMBER</td>
<td>1</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_BCOUNT</td>
<td>2</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_LIST</td>
<td>3</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_MSETUP</td>
<td>4</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_CSETUP</td>
<td>5</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_LTABLE</td>
<td>6</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_LSETUP</td>
<td>7</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_CATALOG</td>
<td>8</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_SYSTEM</td>
<td>9</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_SELF</td>
<td>10</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_MLARGE</td>
<td>11</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_SCONFIG</td>
<td>12</td>
</tr>
<tr>
<td>KI_AGCV_DISPLAY_SERVICE</td>
<td>13</td>
</tr>
</tbody>
</table>

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\_BCOUNT**: Sets displayed page to <BIN COUNT DISPLAY>
- **KI\_AGCV\_DISPLAY\_LIST**: Sets displayed page to <LIST SWEEP DISPLAY>
- **KI\_AGCV\_DISPLAY\_MSETUP**: Sets displayed page to <MEAS SETUP>
- **KI\_AGCV\_DISPLAY\_CSETUP**: Sets displayed page to <CORRECTION>
- **KI\_AGCV\_DISPLAY\_LTABLE**: Sets displayed page to <LIMIT TABLE SETUP>
- **KI\_AGCV\_DISPLAY\_LSETUP**: Sets displayed page to <LIST SWEEP SETUP>
- **KI\_AGCV\_DISPLAY\_CATALOG**: Sets displayed page to <CATALOG>
- **KI\_AGCV\_DISPLAY\_SYSTEM**: Sets displayed page to <SYSTEM INFORMATION>
- **KI\_AGCV\_DISPLAY\_SELF**: Sets displayed 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 Parameters

<table>
<thead>
<tr>
<th>Support</th>
<th>Parameters</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported</td>
<td>instr_id</td>
<td>modifier</td>
</tr>
<tr>
<td>KI_SYSTEM</td>
<td>KI_AVGNUMBER</td>
<td>&lt;value&gt;</td>
</tr>
<tr>
<td></td>
<td>KI_TRIGMODE</td>
<td></td>
</tr>
</tbody>
</table>

**Setmode**

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_MEASX</td>
<td></td>
</tr>
<tr>
<td>KI_INTEGRATE</td>
<td></td>
</tr>
<tr>
<td>KI_AVERAGE</td>
<td></td>
</tr>
<tr>
<td>KI_ABSOLUTE</td>
<td></td>
</tr>
<tr>
<td>KI_NORMAL</td>
<td></td>
</tr>
</tbody>
</table>
### Setmode

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_AVGTIME &lt;value&gt;</td>
<td>(in units of seconds) Time between readings when KI_TRIGMODE is set to KI_AVERAGE.</td>
</tr>
<tr>
<td>No-Op</td>
<td>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.</td>
</tr>
<tr>
<td>KI_HIGH</td>
<td>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.</td>
</tr>
<tr>
<td>KI_LOW</td>
<td></td>
</tr>
<tr>
<td>KI_ON</td>
<td></td>
</tr>
<tr>
<td>KI_OFF</td>
<td></td>
</tr>
<tr>
<td>KI_CC_AUTO &lt;value&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>KI_CC_SRC_DLY &lt;value&gt;</td>
<td>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.</td>
</tr>
<tr>
<td>KI_CC_COMP_DLY &lt;value&gt;</td>
<td></td>
</tr>
<tr>
<td>SMU</td>
<td>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. 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.</td>
</tr>
<tr>
<td>KI_DMM</td>
<td>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.</td>
</tr>
<tr>
<td>KI_IMTR</td>
<td>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.</td>
</tr>
<tr>
<td>KI_S400</td>
<td>Sets the SMU to use ranges equivalent to the Model S400.</td>
</tr>
<tr>
<td>KI_TRGME</td>
<td></td>
</tr>
</tbody>
</table>
Setmode

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>KI_ELECTROMETER</td>
<td>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.</td>
</tr>
<tr>
<td>KI_LIM_INDCTR</td>
<td>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.</td>
</tr>
<tr>
<td>KI_LIM_MODE</td>
<td>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.</td>
</tr>
<tr>
<td>KI_RANGE_DELAY</td>
<td>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.</td>
</tr>
<tr>
<td>KI_RANGE_SETTL</td>
<td>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</td>
</tr>
<tr>
<td>KI_VMTR</td>
<td>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.</td>
</tr>
<tr>
<td>KI_S400</td>
<td>Sets the SMU to use ranges equivalent to the Model S400.</td>
</tr>
<tr>
<td>KI_DMM</td>
<td>Sets the SMU to use ranges equivalent to a DMM. Provides a low impedance, fast measurement. Used for low voltage applications.</td>
</tr>
<tr>
<td>KI_ELECTROMETER</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 disturbances.

Example: status = setmode("CVU1", KI_CVU_OPEN_COMPENSATE, isCmpstOpen=0)
smeasz_sweepv

Purpose: Performs and returns DC 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, K1_CVU_SPEED_FAST,-3,3,10,0.01)
```

srangei/srangev

Purpose: Selects the current/voltage source range and prevents the selected instrument from autoranging. When you select a range, the time required for autoranging is eliminated.

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

trigXg, trigXl: Trigger greater than, less than

Purpose: Monitors for a predetermined level of voltage, current, or time.

Format:
```
trigig(SMUX, value);
trigil(SMUX, value);
trigtg(ntimer[Y], value);
trigtl(ntimer[Y], value);
trigvg(SMUX, value);
trigvl(SMUX, value);
```

value – The voltage, current, or time specified as the trigger point. This trigger point value is considered to be reached when:

• The measured value is equal to or greater than the value argument of trigXg

• The measured value is less than the value argument of trigXl.

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 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

ACS LPT using example: `vgsid1`

```python
#outputlist=GateV, DrainI, Time#
from ACS_PostData import *
from ACSLPT import *
from ptmlpt.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 INVAL_PARAM
    if GateVStop < -200 or GateVStop > 200:
        return INVAL_PARAM
    if numberofpoint < 1 or numberofpoint > 4096:
        return INVAL_PARAM
    if SweepDelay < 0 or SweepDelay > 100:
        return INVAL_PARAM
    if DrainV < -200 or DrainV > 200:
        return INVAL_PARAM
    if SourceV < -200 or SourceV > 200:
        return INVAL_PARAM
    if BulkV < -200 or BulkV > 200:
        return INVAL_PARAM
    if RangeDrainI < 1 or RangeDrainI > 12:
        return INVAL_PARAM
    if ComplianceDrainI < -0.1 or ComplianceDrainI > 0.1:
        return INVAL_PARAM
    # Switch Matrix connection
    clrcon()
    if GatePin > 0:
        conpin(GateSMU, GatePin)
    if DrainPin > 0:
        conpin(DrainSMU, DrainPin)
    if SourcePin > 0:
        conpin(SourceSMU, SourcePin)
    if BulkPin > 0:
        conpin(BulkSMU, BulkPin)
```
# Set the SMUs range
rangei(GateSMU, 0.1)
rangei(BulkSMU, 0.1)
rangei(SourceSMU, 0.1)
setauto(DrainSMU)
limiti(DrainSMU, ComplianceDrainI)

# best fix for voltage range
if fabs(SourceV) < 0.2:
    rangev(SourceSMU, 0.2)
elif fabs(SourceV) < 2:
    rangev(SourceSMU, 2)
elif fabs(SourceV) < 20:
    rangev(SourceSMU, 20)
else:
    rangev(SourceSMU, 200)
if fabs(BulkV) < 0.2:
    rangev(BulkSMU, 0.2)
elif fabs(BulkV) < 2:
    rangev(BulkSMU, 2)
elif fabs(BulkV) < 20:
    rangev(BulkSMU, 20)
else:
    rangev(BulkSMU, 200)
if fabs(DrainV) < 0.2:
    rangev(DrainSMU, 0.2)
elif fabs(DrainV) < 2:
    rangev(DrainSMU, 2)
elif fabs(DrainV) < 20:
    rangev(DrainSMU, 20)
else:
    rangev(DrainSMU, 200)
if fabs(GateVStart) > fabs(GateVStop):
    temp = fabs(GateVStart)
else:
    temp = fabs(GateVStop)
if temp < 0.2:
    rangev(GateSMU, 0.2)
elif temp < 2:
    rangev(GateSMU, 2)
elif temp < 20:
    rangev(GateSMU, 20)
else:
    rangev(GateSMU, 200)
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:                   #limited auto 10nA
    lorangei(DrainSMU, 1e-8)
elif RangeDrainI == 6:                   #limited auto 100nA
    lorangei(DrainSMU, 1e-7)
elif RangeDrainI == 7:                   #limited auto 1μA
    lorangei(DrainSMU, 1e-6)
elif RangeDrainI == 8:                   # limited auto 10μA
    lorangei(DrainSMU, 1e-5)
elif RangeDrainI == 9:                   # limited auto 100μA
    lorangei(DrainSMU, 1e-4)
elif RangeDrainI == 10:                  # limited auto 1mA
    lorangei(DrainSMU, 1e-3)
elif RangeDrainI == 11:                  # limited auto 10mA
    lorangei(DrainSMU, 1e-2)
elif RangeDrainI == 12:                 # limited auto 100mA
    lorangei(DrainSMU, 0.1)
else:                                  #limited auto 10mA
    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

vgsid1(DrainSMU, DrainPin, GateSMU, GatePin, SourceSMU, SourcePin, BulkSMU,
       BulkPin, GateVStart, GateVStop, numberofpoint, SweepDelay, DrainV, SourceV,
       BulkV, RangeDrainI, ComplianceDrainI, StoponCompliance, NPLC)
Series 2600A library introduction

The ACS software has a test library of commands (26Library), that is a library for the Series 2600A SourceMeter® instruments. The library path is \ACS\library\26Library. It includes device test libraries and a WLR library. The tables in this section summarize all the test modules in the device test and WLR libraries.

### BJT Test Library

<table>
<thead>
<tr>
<th>Module</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Three_term_BJT_BVCBO</td>
<td>Three_term_BJT_BVCEI</td>
</tr>
<tr>
<td>Three_term_BJT_BVCES</td>
<td>Three_term_BJT_BVEBO</td>
</tr>
<tr>
<td>Three_term_BJT_BVECO</td>
<td>Three_term_BJT_HFE_sw</td>
</tr>
<tr>
<td>Three_term_BJT_ICEO</td>
<td>Three_term_BJT_IBEO</td>
</tr>
<tr>
<td>Three_term_BJT_IBVBE</td>
<td>Three_term_BJT_ICBO</td>
</tr>
<tr>
<td>Three_term_BJT_ICES</td>
<td>Three_term_BJT_ICEV</td>
</tr>
<tr>
<td>Three_term_BJT_icvce_biasIB</td>
<td>Three_term_BJT_icvce_biasVB</td>
</tr>
<tr>
<td>Three_term_BJT_icvce_stepvb</td>
<td>Three_term_BJT_IEBO</td>
</tr>
<tr>
<td>Three_term_BJT_ieveb</td>
<td>Three_term_BJT_VBCO</td>
</tr>
<tr>
<td>Three_term_BJT_BVCES</td>
<td>Three_term_BJT_BVCEV</td>
</tr>
<tr>
<td>Three_term_BJT_BVCEI</td>
<td>Three_term_BJT_HFE_tral</td>
</tr>
<tr>
<td>Three_term_BJT_BVECO</td>
<td>Three_term_BJT_IBicvbe</td>
</tr>
<tr>
<td>Three_term_BJT_ICEO</td>
<td>Three_term_BJT_ICEO</td>
</tr>
<tr>
<td>Three_term_BJT_IBVBE</td>
<td>Three_term_BJT_IVCEO</td>
</tr>
<tr>
<td>Three_term_BJT_ICES</td>
<td>Three_term_BJT_icvcb</td>
</tr>
<tr>
<td>Three_term_BJT_icvce_biasIB</td>
<td>Three_term_BJT_icvce_stepib</td>
</tr>
<tr>
<td>Three_term_BJT_icvce_stepvb</td>
<td>Three_term_BJT_IECO</td>
</tr>
<tr>
<td>Three_term_BJT_ieveb</td>
<td>Three_term_BJT_VBCO</td>
</tr>
</tbody>
</table>

### MOSFET Test Library

<table>
<thead>
<tr>
<th>Module</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Four_term_MOSFET_BVDSS</td>
<td>Four_term_MOSFET_BVDSV</td>
</tr>
<tr>
<td>Four_term_MOSFET_BVGDSS</td>
<td>Four_term_MOSFET_BVGDOS</td>
</tr>
<tr>
<td>Four_term_MOSFET_IDS_ISD</td>
<td>Four_term_MOSFET_idvd</td>
</tr>
<tr>
<td>Four_term_MOSFET_idvg</td>
<td>Four_term_MOSFET_idvg_vd</td>
</tr>
<tr>
<td>Four_term_MOSFET_IGL</td>
<td>Four_term_MOSFET_igvg</td>
</tr>
<tr>
<td>Four_term_MOSFET_isubvg</td>
<td>Four_term_MOSFET_Vth_ci</td>
</tr>
<tr>
<td>Four_term_MOSFET_BVDSS</td>
<td>Four_term_MOSFET_BVDSV</td>
</tr>
<tr>
<td>Four_term_MOSFET_BVGDSS</td>
<td>Four_term_MOSFET_BVGDOS</td>
</tr>
<tr>
<td>Four_term_MOSFET_IDS_ISD</td>
<td>Four_term_MOSFET_idvd</td>
</tr>
<tr>
<td>Four_term_MOSFET_idvg</td>
<td>Four_term_MOSFET_idvg_vd</td>
</tr>
<tr>
<td>Four_term_MOSFET_IGL</td>
<td>Four_term_MOSFET_igvg</td>
</tr>
<tr>
<td>Four_term_MOSFET_isubvg</td>
<td>Four_term_MOSFET_Vth_ci</td>
</tr>
<tr>
<td>Four_term_MOSFET_BVDSS</td>
<td>Four_term_MOSFET_BVDSV</td>
</tr>
<tr>
<td>Four_term_MOSFET_BVGDSS</td>
<td>Four_term_MOSFET_BVGDOS</td>
</tr>
<tr>
<td>Four_term_MOSFET_IDS_ISD</td>
<td>Four_term_MOSFET_idvd</td>
</tr>
<tr>
<td>Four_term_MOSFET_idvg</td>
<td>Four_term_MOSFET_idvg_vd</td>
</tr>
<tr>
<td>Four_term_MOSFET_IGL</td>
<td>Four_term_MOSFET_igvg</td>
</tr>
<tr>
<td>Four_term_MOSFET_isubvg</td>
<td>Four_term_MOSFET_Vth_ci</td>
</tr>
</tbody>
</table>
The next table summarizes all test modules in the WLR library. More detailed descriptions of each module follow the tables.

### WLR Test Library

<table>
<thead>
<tr>
<th>Module</th>
<th>Module</th>
<th>Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>HCI</td>
<td>NBTI</td>
<td>NBTI_on_the_fly</td>
</tr>
<tr>
<td>TDDDB_CCS</td>
<td>TDDDB_per_pin</td>
<td>qbd_rmpj</td>
</tr>
<tr>
<td>qbd_rmpv</td>
<td>Em_iso_test</td>
<td>NBTI_meas</td>
</tr>
</tbody>
</table>

### Series 2600A general notes

#### Create a library without Script Editor

You can use Test Script Language on the Keithley Instruments Series 2600A 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 2600A. Refer to the LPT Library Reference in the Series 2600A 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:

1. 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:
2. The types of input variables must be:
   - instid (SMU input)
   - string
   - double
   - integer
   - table

3. You can set a default value for every input variable. You can also set the input range for double and integer-type input variables. 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

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

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

**NOTE**
Refer to the following directory for examples: \ACS \Library\26Library\WLR folder.

Create a library using Script Editor (recommended)

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 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 2600A instrument to create test script files, based on the Series 2600A 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 2600A

DUT: Three term BJT

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

Pin Connections: Open emitter, apply desired current to collector, base connects to ground.

Intended results: Get the collector-base breakdown voltage (BVCBO).

**Three_term_BJT_BVCEI**

Module Name: Three_term_BJT_BVCEI

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test collector-emitter breakdown voltage of BJT, with base-forced current bias.

Pin Connections: Apply the desired current to collector, set base bias current (emitters usually connect to ground).

Intended results: Get collector-emitter breakdown voltage.

**Three_term_BJT_BVCEO**

Module Name: Three_term_BJT_BVCEO

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

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

Pin Connections: Open base, apply desired current to collector, 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 2600A
DUT: Three term BJT
Function: Test the collector-emitter breakdown voltage of BJT, with base-emitter shorted.
Pin Connections: Apply desired current to collector. Base and emitter connect to ground.
Intended results: Get collector-emitter breakdown voltage.

Three_term_BJT_BVCEV
Module Name: Three_term_BJT_BVCEV
Instrument: Keithley Instruments Series 2600A
DUT: Three term BJT
Function: Test the collector-emitter breakdown voltage, with base forced as voltage bias.
Pin Connections: Apply desired current to collector, set base as bias voltage, connect emitter to ground.
Intended results: Get collector-emitter breakdown voltage.

Three_term_BJT_BVEBO
Module Name: Three_term_BJT_BVEBO
Instrument: Keithley Instruments Series 2600A
DUT: Three term BJT
Function: Test the emitter-base breakdown voltage of BJT with collector open.
Pin Connections: Open collector, set emitter at desired current, connect 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 2600A
DUT: Three term BJT
Function: Test the emitter-collector breakdown voltage of BJT, with base open.
Pin Connections: Open base, apply the desired current to emitter, connect collector to ground.
Intended results: Get the BVECO.

**Three_term_BJT_HFE_sw**

Module Name: Three_term_BJT_HFE_sw

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test the HFE (DC current gain) of BJT, with collector voltage sweep.

Pin Connections: Sharing emitter connection, apply sweep voltage on collector, apply bias voltage on base, emitter connect to ground, if not apply bias voltage on it.

Use this technique:

1. Force CollectorV sweep.
2. Measure Ib and Ic.
3. Check for measurement problems.
4. Calculate HFE= Ic/Ib.

Intended results: Get collector current, base current, and DC current gain according to collector sweep voltage.

**Three_term_BJT_HFE_tral**

Module name: Three_term_HFE_tral

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test the DC current gain (HFE) of BJT by traditional test method.

Pin Connections: Sharing emitter connection, apply a desired voltage on collector and apply current on base. Emitter usually connected to ground, if not, specified voltage applied on it (see next Figure).
Figure 24: 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 point at target collector current, record base current.
4. Calculate $HFE = \frac{IC_{tar}}{IB}$.

Intended Results: Get the HFE.

**Three_term_BJT_IBCO**

Module Name: Three_term_BJT_IBCO

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test base-collector current with emitter open.

Pin Connections: Open emitter, apply voltage on base, and apply voltage to collector (if not connected to ground).

Intended results: Get base-collector current.

**Three_term_BJT_IBEO**

Module Name: Three_term_BJT_IBEO

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test the base-emitter current with collector open
Pin Connections: Sharing base connection, apply sweep voltage on collector, apply bias voltage on emitter. The base is usually connected to ground, but can be set to a desired bias voltage.

**Three_term_BJT_ibicvbe**

Module Name: Three_term_BJT_ibicvbe

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test base current and collector current of BJT by specified base voltage sweep.

Pin Connections: Sharing emitter connection, apply sweep voltage on base, apply bias voltage on collector. 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 Ib-Vbe and Ic-Vbe curve.
3. Get the gummel plot if the axis properties of data plot have changed (logarithm instead of right-angle coordinate).

**Three_term_BJT_icvbe**

Module Name: Three_term_BJT_icvbe

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test base current of BJT by specified base voltage sweep.

Pin Connections: Sharing emitter connection, apply sweep voltage on base, apply bias voltage on collector. Emitter is usually connected to ground, but can be set to the desired bias voltage.

Intended results:

1. Get measured base current according to base voltage sweep.
2. Get Ib-Vbe curve.

**Three_term_BJT_ICBO**

Module Name: Three_term_BJT_ICBO

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

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

Pin Connections: Open emitter, apply a desired voltage to collector, base 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 term BJT
Function: Test the collector-emitter cut off current with base open.
Pin Connections: Open base, apply a desired voltage to collector, connect emitter to ground.
Intended results: Get collector-emitter to cut off current.

Three_term_BJTICES
Module Name: Three_term_BJTICES
Instrument: Keithley Instruments Series 2600A
DUT: Three term BJT
Function: Test the collector-emitter cut-off current with base-emitter shorted.
Pin Connections: Apply a desired voltage to collector; connect emitter and base to ground.
Intended results: Get collector-emitter to cut off current.

Three_term_BJT_ICEV
Module Name: Three_term_BJT_ICEV
Instrument: Keithley Instruments Series 2600A
DUT: Three term BJT
Function: Test the collector-emitter cut off current with base voltage bias.
Pin Connections: Apply desired voltage to collector, apply voltage bias on base, connect emitter to ground.
Intended results: Get collector-emitter to cut off current.

Three_term_BJT_icvcb
Module Name: Three_term_BJT/icvcb
Instrument: Keithley Instruments Series 2600A
DUT: Three term BJT
Function: Test collector current of BJT by specified collector voltage sweep.
Pin Connections: Sharing emitter connection, apply sweep voltage on collector, apply bias voltage on base. Emitter is usually connected to ground, but can be set to the desired bias voltage.

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

**Three_term_BJT_icvce_biasIB**

Module Name: Three_term_BJT_icvce_biasIB

Instrument: Keithley Instruments Model 4200

DUT: Three term BJT

Function: Test a series of Ic-Vce curves of BJT when base current.

Pin Connections: Sharing emitter connection (connect emitter to ground), step base current, sweep collector voltage.

Intended results: Get measured collector current according to base step current and collector sweep voltage.

**Three_term_BJT_icvce_biasVB**

Module Name: Three_term_BJT_icvce_biasVB

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test a series of Ic-Vce curves of BJT when base voltage.

Pin Connections: Sharing emitter connection (connect emitter to ground), step base voltage, sweep collector voltage.

Intended results: Get measured collector current according to 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 term BJT

Function: Test a series of Ic-Vce curves of BJT when stepping base current.

Pin Connections: Sharing emitter connection (connect emitter to ground), step base current, sweep collector voltage.

Intended results:
1. Get measured collector current according to base step current and collector sweep voltage.
2. Get a series of Ic-Vce curves by base current step.
Three_term_BJT_icvce_stepvb

Module Name: Three_term_BJT_icvce_stepvb

Instrument: Keithley Instruments Model 4200

DUT: Three term BJT

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

Pin Connections: Sharing emitter connection (connect emitter to ground), step base voltage, sweep collector voltage.

Intended results:
1. Get measured collector current according to base step and collector sweep voltage.
2. Get a series of Ic-Vce curves by base voltage step.

Three_term_BJT_IEBO

Module Name: Three_term_BJT_IEBO

Instrument: Keithley Instruments Model 4200

DUT: Three term BJT

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

Pin Connections: Open collector, apply desired voltage to the emitter, base connects to ground.

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

Three_term_BJT_IECO

Module Name: Three_term_BJT_IECO

Instrument: Keithley Instruments Model 4200

DUT: Three term BJT

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

Pin Connections: Open base, apply desired voltage to emitter. Collector is usually connected to ground if voltage is not applied.

Intended results: Get emitter-collector current.

Three_term_BJT_iieveb

Module Name: Three_term_BJT_iieveb

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT
Function: Test emitter current of BJT by specified emitter voltage sweep.

Pin Connections: Sharing base connection, apply sweep voltage on emitter, apply bias voltage on collector. Connect base to ground if voltage is not applied.

Intended results:
1. Get measured emitter current according to emitter voltage sweep.
2. Get Ie-Veb curves.

**Three_term_BJT_VBCO**

Module Name: Three_term_BJT_VBCO

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

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

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

Intended results: Get base-collector voltage.

**Three_term_BJT_VCE**

Module Name: Three_term_BJT_VCE

Instrument: Keithley Instruments Series 2600A

DUT: Three term BJT

Function: Test the collector-emitter voltage BJT.

Pin Connections: Apply voltage on base, set collector current to desired level, connect 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 gmlin, gmsat, idvd, idvg, igvg, Vtci, and Vtex. The 26library is used with a Series 2600A to create test script files, based on the Series 2600A 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 2600A
DUT: Four term MOSFET
Function: Used to test Drain-Source breakdown voltage of MOSFET, with Gate-Source shorted.
Pin Connections: Apply a breakdown current on Drain. Connect bulk to ground, if not, force 0 voltage. Gate, source connected to ground; otherwise force 0 voltage.
Intended results: Get breakdown voltage between Drain and Source with Gate-Source shorted.

Four_term_MOSFET_BVDSV
Module Name: Four_term_MOSFET_BVDSV
Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET
Function: Used to test Drain-Source breakdown voltage of MOSFET, with Gate biased.
Pin Connections: Source and Bulk connect to ground. Gate biased. Apply a breakdown current on Drain.
Intended results: Get breakdown voltage between Drain and Source with Gate biased.

Four_term_MOSFET_BVGSO
Module Name: Four_term_MOSFET_BVGSO
Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET
Function: Used to test Gate-Source breakdown voltage of MOSFET, with Drain opened.
Pin Connections: Open Drain. Connect Bulk and Source to ground. Apply a breakdown current on Gate.
Intended results: Get breakdown voltage between Gate and Source with Drain opened.

Four_term_MOSFET_BVGDS
Module Name: Four_term_MOSFET_BVGDS
Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET
Function: Used to test Gate-Drain breakdown voltage of MOSFET, with Source-Drain shorted.
Pin Connections: Connect Source, Drain to ground. Apply a breakdown current on Gate.

Intended results: Get breakdown voltage between Gate and Drain with Source-Drain shorted.

**Four_term_MOSFET_BVGDO**

Module Name: Four_term_MOSFET_BVGDO

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test Gate-Drain breakdown voltage of MOSFET, with Source opened.

Pin Connections: Open source. Connect Bulk and drain to ground. Apply a breakdown current on Gate.

Intended results: Get breakdown voltage between Gate and Drain when Source is open.

**Four_term_MOSFET_IDL**

Module Name: Four_term_MOSFET_IDL

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to measure Drain leakage current with Gate-Source shorted.

Pin Connections: Short Gate and Source. Apply a voltage on Drain. Bulk, Gate and Source connected to ground.

Intended results: Get Drain leakage current with Gate-Source shorted.

**Four_term_MOSFET_IDS_ISD**

Module Name: Four_term_MOSFET_IDS_ISD

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to measure Drain-source and Source-Drain current with Gate biased.

Pin Connections: Apply voltage separately on Gate, Source, and Drain. Bulk is usually connected to ground, but can be set with a desired bias voltage.

Intended results: Get measured Drain-Source and Source-Drain current with Gate biased.

**Four_term_MOSFET_idvd**

Module Name: Four_term_MOSFET_idvd

Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET

Function: Used to test Drain current at specified Drain voltage sweep.

Pin Connections: Gate biased. Drain sweep. Connect Bulk and Source to ground if voltage is not applied.

Intended results:

Get measured Drain current at specified Drain voltage sweep.

Get Drain current versus Drain voltage curve.

Four_term_MOSFET_idvd_vg

Module Name: Four_term_MOSFET_idvd_vg

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test a series of Id_Vd curves for four-terminal MOSFET, which performs on the Series 2600A.

Pin Connections: Sweep Drain, Step gate. Connect Bulk and Source to ground if voltage is not applied.

Intended results:

Get measured Drain current at specified Drain voltage sweep.

Get a series of Drain current versus Drain voltage curve.

Four_term_MOSFET_idvg

Module Name: Four_term_MOSFET_idvg

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test Drain current at specified Gate voltage sweep.

Pin Connections: Drain biased, Gate sweep. Connect Bulk and Source to ground if voltage is not applied.

Intended results:

Get measured Drain current at Gate voltage sweep.

Get Drain current versus Gate voltage curve.

Four_term_MOSFET_idvg_vd
Module Name: Four_TERM_MOSFET_idvg_vd

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test Drain current at specified Gate voltage sweep, with Drain step.

Pin Connections: Drain step, Gate sweep. Connect Bulk and Source to ground if voltage is not applied.

Intended results:
Get measured Drain current at Gate voltage sweep.
Get a series of Drain currents versus Gate voltage curve.

Four_TERM_MOSFET_idvg_vsub

Module Name: Four_TERM_MOSFET_idvg_vsub

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test Drain current at specified Gate voltage sweep, with Bulk step.

Pin Connections: Bulk step. Gate sweep. Drain biased. Connect Source to ground if voltage is not applied.

Intended results:
Get measured Drain current at Gate voltage sweep.
Get a series of Drain currents versus Gate voltage curve.

Four_TERM_MOSFET_IGL

Module Name: Four_TERM_MOSFET_IGL

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to measure gate leakage current, with Source-Drain short.

Pin Connections: Apply a voltage on Gate. Source, Drain, and Bulk connected to ground. Intended results: Get gate leakage current when short Source and Drain

Four_TERM_MOSFET_igvg

Module Name: Four_TERM_MOSFET_igvg

Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET

Function: Used to test Gate current at specified Gate voltage sweep when Drain biased.

Pin Connections: Drain biased, Gate sweep. Connect Bulk and Source to ground (if not, apply desired voltage).

Intended results:

Get measured Gate current at Gate voltage sweep.
Get Gate current versus Gate voltage curve.

Four_term_MOSFET_ISL
Module Name: Four_term_MOSFET_ISL
Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to measure Source leakage current when Gate-Drain short.

Pin Connections: Apply a voltage on Source. Bulk, Gate, and Drain connected to ground.

Intended results: Get Source leakage current when Gate-Drain short.

Four_term_MOSFET_isubvg
Module Name: Four_term_MOSFET_isubvg
Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

Function: Used to test Bulk current at specified Gate voltage sweep.

Pin Connections: Drain and Bulk biased. Gate sweep. Source connected to ground if voltage is not applied.

Intended results:

Get measured Bulk current at Gate voltage sweep.
Get Bulk current versus Gate voltage curve.
Get maximal Bulk current and corresponding Gate voltage.

Four_term_MOSFET_Vth_ci
Module Name: Four_term_MOSFET_Vth_ci
Instrument: Keithley Instruments Series 2600A
DUT: Four term MOSFET

Function: Used to get the constant current threshold voltage of MOSFET.

Pin Connections: Drain biased. Gate sweep. Input source and bulk voltage when needed; they 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:

\[ V_{th_{ci}} = V_{GS} (@ID=1\mu A/W/L) \quad -- \quad \text{NMOS} \]
\[ V_{th_{ci}} = V_{GS} (@ID=-0.025\mu A/W/L) \quad -- \quad \text{PMOS} \]

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 is the sign of being near threshold, then search the Gate voltage to make the Drain current equals \( I_{d_{tar}} \).

**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, \( V_{DS_{lin}} = -0.1 \text{ V (@VDD=5V)} \); for NMOS, \( V_{DS_{lin}}=0.1\text{V(@VDD=5V)} \).

Intended results:

Get the constant-current threshold voltage.

Get Drain current versus Gate voltage curve.

**Four_term_MOSFET_Vth_ex**

Module Name: Four_term_MOSFET_Vth_ex

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

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

Pin Connections: Drain biased. Sweep gate voltage. Input Source and Bulk voltage when needed. Usually, they are connected to ground for NMOS, and connected to the normal power supply voltage (VDD) for PMOS.

Technique: The threshold voltage is extrapolated from measurement of maximum slope (Gmmax) of the ID-VGS curve, as described below:

\[ V_{th_{ex}} = V_{GS} (@G_{m_{max}}) - I_{D}(@G_{m_{max}}) / G_{m_{max}} \]
Where: \( V_{GS}(G_{m_{max}}) \) is the gate voltage at the point of the maximum slope of the ID-VGS curve; \( I_D(G_{m_{max}}) \) is the drain current at the point of the maximum slope of the ID-VGS curve; \( G_{m_{max}} \) 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, \( V_{DS \_lin} = -0.1 \ V(\@V_{DD} = 5\ V) \); for NMOS, \( V_{DS \_lin} = 0.1V(\@V_{DD} = 5\ V) \).

Intended results:

Get measured Drain current at Gate voltage sweep.

Extract trans_conductance\((G_m)\), get maximum transconducntance \((G_{m_{max}})\).

Get extracted threshold voltage \((V_{th \_ex})\).

Get Drain current versus Gate voltage curve.

Get \( G_m \) versus Drain current or \( G_m \) versus Gate voltage curve.

**Four_term_MOSFET_Vth_lsql**

Module Name: Four_term_MOSFET_Vth_lsql

Instrument: Keithley Instruments Series 2600A

DUT: Four term MOSFET

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

Pin Connections: Drain biased. Sweep Gate voltage. Input Source and Bulk voltage when needed. Usually, they are connected to ground for NMOS, and connected to the normal power supply voltage \((V_{DD})\) for PMOS.

Technique: The threshold voltage is extrapolated from measurement of maximum slope \((G_{m_{max}})\) of the ID-VGS curve, as described below:

\[
V_{th \_ex} = V_{GS}(G_{m_{max}}) - I_D(G_{m_{max}})/G_{m_{max}}.
\]

Where: \( V_{GS}(G_{m_{max}}) \) is the gate voltage at the point of the maximum slope of the ID-VGS curve; \( I_D(G_{m_{max}}) \) is the drain current at the point of the maximum slope of the ID-VGS curve; \( G_{m_{max}} \) 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, \( V_{DS \_lin} = -0.1 \ V(\@V_{DD} = 5\ V) \); for NMOS, \( V_{DS \_lin} = 0.1V(\@V_{DD} = 5\ V) \).
Intended results:

Get measured Drain current at Gate voltage sweep.

Extract trans-conductance (Gm), get maximum trans-conductance (Gmmax).

Get extracted threshold voltage (Vth_ex).

Get Drain current versus Gate voltage curve.

Get Gm versus Drain current or Gm versus Gate voltage curve.

**Four_term_MOSFET_Vth_sense**

Module Name: Four_term_MOSFET_Vth_sense

Instrument: Keithley Instruments Series 2600A in sense mode

DUT: Four term MOSFET

Function: Used to get self-biasing threshold voltage (Vth_sel) for four-terminal MOSFET (only Keithley Instruments provides a two-channel SMU with this capability).

Pin connections (see next Figure).

Use two SMUs: SMU_A and SMU_B. SMU_A is (in sense mode).

Bulk, source, HI terminal of SMU_B, and sense_HI terminal of SMU_A are connected together. Gate, LO terminal of SMU_B and HI terminal of SMU_A are connected together.

Drain, sense_LO, and LO terminal of SMU_A are connected together.

**Figure 25: Four_term_MOSFET_Vth_sense pin connections**
Technique:
1. Set SMU_B to desired drain current level (IDS).
2. Set SMU_A to desired VDS level.
3. Measure VG using SMU_B.
4. Ensure Vth = VG.

**NOTE**
When in sense mode, SMU_A will automatically vary the voltage on the gate until VDS is equal to the desired level; at that time, measure gate voltage. The threshold voltage should equal the measured VG.

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 2600A to create test script files based on the Series 2600A 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 2600A

DUT: Diode

Function: Calculates the Dynamic Impedance based on 2 forward voltage or 2 reverse voltage measurements. DynamicZ = (v2 - v1) / (I2 - I1)

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

Intended results: Get Dynamic impedance.

**Diode_Ifd_Vfd**

Module Name: Diode_Ifd_Vfd

Instrument: Keithley Instruments Series 2600A

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

Pin Connections: P terminal forced forward voltage, N terminal is grounded.

Intended results: Get forward current.

Diode_Ifd_Vfd_vsweep
Module Name: Diode_Ifd_Vfd_vsweep
Instrument: Keithley Instruments Series 2600A
DUT: Diode

Function: Test forward current with forward voltage sweep, to indicate forward I-V characteristic of diode.

Pin Connections: Apply a forward sweep voltage to terminal P; connect N terminal to ground.

Intended results: Forward voltage according to forward current sweep.

Diode_Ileakage_Vrd
Module Name: Diode_Ileakage_Vrd
Instrument: Keithley Instruments Series 2600A
DUT: Diode

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

Pin Connections: Apply a forced reverse voltage or zero voltage to the N terminal; connect P terminal to ground.

Intended results: Reverse leakage current.

Diode_Ird_Vrd_vsweep
Module Name: Diode_Ird_Vrd_vsweep
Instrument: Keithley Instruments Series 2600A
DUT: Diode

Function: Test reverse current with reverse voltage sweep 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 current at each reverse voltage sweep point.

Diode_Vbr_Ird
Module Name: Diode_Vbr_Ird
Instrument: Keithley Instruments Series 2600A

DUT: Diode

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

Pin Connections: Force a reverse current to the N terminal; connect P terminal to ground.

Intended results: Breakdown voltage.

Diode_Vfd_Ifd

Module Name: Diode_Vfd_Ifd

Instrument: Keithley Instruments Series 2600A

DUT: Diode

Function: Test the Forward Voltage of a diode.

Pin Connections: Uses one SMU to force forward current, while the other terminal is grounded. Forward voltage is measured at the current.

Intended results: Forward voltage.

Diode_Vrd_Ird

Module Name: Diode_Vrd_Ird

Instrument: Keithley Instruments Series 2600A

DUT: Diode

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

Pin Connections: Force a reverse current to terminal N; connect the P terminal to ground.

Intended results: Reverse voltage.

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 2600A to create test script files based on Series 2600A 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 2600A

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 2600A

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 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 2600A instruments. HCI, TDDB, and two NBTI tests are available in this library. They are test script files, based on the Series 2600A LPT library.

HCI

Pin Connections: A 4-terminal MOSFET is used in this test. Source and Bulk can be connected to GROUND manually; 2 to 4 SMUs are needed. The test consists of two parts: A test part and a stress part.

For the STRESS part, 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 part of the test.

For the TEST part, the following tests 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 drain current ‘Id_sat’ test. If saturate drain voltage ‘Vd_sat’ is not empty, measure: Id_sat@Vd=Vd_sat, Vg = Vd_sat.

• Drain leakage current ‘Id_leak’ test. If drain leakage voltage ‘Vd_leak’ is not empty, measure: Id_leak@Vd = Vd_leak, Vg = Vb.

• Gate leakage current ‘Ig_leak’ test. If gate leakage voltage ‘Vg_leak’ is not nil, measure: Ig_leak@Vg=Vg_leak, 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


‘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, ld_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 [0,2] --maximum time for the test. Not in use when t_mode is

integer npdec_delta=3 in [0,2] --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,]  -- compliance on source 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,]  -- compliance on source 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,]  -- compliance on gate during test (Unit: A)
double Vd_lin=0.1  -- drain voltage in linear district (Unit: V)
double D_comp=0.1 in [0,]  -- compliance on drain during test (Unit: A)
double Id_target=1e-4  -- if not nil, Vtci will be calculated and output instead of Vtex
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])
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 26: GUI for HCI
• Terminal Stress Setup: Set the SMUs for each terminal, set voltage and corresponding compliances during the STRESS and TEST. If 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 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
```
TDDB_CCS

This function is used to perform the Constant Current Time Dependent Dielectric Breakdown test. Up to 4 SMUs are supported. Only voltage is measured. HBD happens when:

If the Vg is below breakdown voltage (abs(Vg)<abs(Vmin)).

If the Vg falls dramatically (abs(Vg[now]) <= HBDL * abs(Vg[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:**

- `sample_interval` = 1 in [0, ) --time between sample (Unit:s)
- `HBDL` = 0.6 in [0, 0.999] --limit of hard BD. when Vg[now]=Vg[prev]*HBDL then abort.
- `V_min` = 0.06 in [0, 200] --minimum voltage
- `time_max` = nil in [0, )/nil --max time of experiment. if 'nil' appears, test until BD
- `holdtime` = 0 in [0, ) --time before stress begin (Unit:s)
- `myPLC` = 1 in [0.001, 25] --PLC setting
- `smu_1` = 1 in [0, 1, 2..64] --maximum 4 smus are supported. if not input '0'
- `comp1` = 2 --compliance of corresponding smu (Unit:A for current; V for voltage)
- `stress1` = 1e-6 --stress value required on the smu (Unit:A for current; V for voltage)
- `meas1` = 1 in [0, 1] --stress no measurement
- `smu_2` = 2 in [0, 1, 2..64] --maximum 4 smus are supported. if not input '0'
- `comp2` = 0.1 current; V for voltage --compliance of corresponding smu (Unit:A for current; V for voltage)
- `stress2` = 0 current; V for voltage --stress value required on the smu (Unit:A for current; V for voltage)
- `meas2` = 1 in [0, 1] --1: current stress and make measurement 0: voltage
integer meas2=0 in [0, 1] stress no measurement
--1: current stress and make measurement 0: voltage
integer smu_3=0 in[0, 1, 2...64] --maximum 4 smus are supported. if not input '0'
double comp3=nil current;V for voltage) --compliance of corresponding smu (Unit:A for
double stress3=nil for voltage) --stress value required on the smu (Unit:A for current; V
integer meas3=nil 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 current;V for voltage) --compliance of corresponding smu (Unit:A for
double stress4=nil for voltage) --stress value required on the smu (Unit:A for current; V
integer meas4=nil voltage stress no measurement
--1: current stress and make measurement 0:

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

**Figure 27: GUI for TDDB_CCS**

**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 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**

```python
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=nil
local stress3=nil
local meas3=nil
local smu_4=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 test. Up to 4 SMUs are supported. Only voltage is forced and only current is measured.

1. If the breakdown mode is 0, only Hard BreakDown (HBD) will be monitored. If breakdown mode is 1, Soft BreakDown (SBD) will also be monitored.
2. HBD occurs when Ig[now] = HBDL*Ig[prev].
3. To evaluate SBD, calculate the noise of gate current (Inoi) from the formula listed in JESD92; the base noise (Inoi_base) is calculated with the Inoi average value (AVL) and base number (bas_num). When Inoi_base is set, SBD occurs if several sequential Inois meet the following condition: Inoi[now] = SBDL*Inoi_base.
4. If the DUT is a MOSFET, set the SMUs that do not need to be measured to 0 (meas = 0).
5. Intended outputs: time, Ig, Ig_noise (when SBD is required), and BreakDown_type of SMUs requiring measurement.

Syntax:

```plaintext
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 Ig[now] >= Ig[prev]*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 Inoi[now] = Inoi[base]*SBDL, then abort

integer AVL=10
---Standard when calculating base noise current; if Inoi[now]<=AVL*Inoi_base, Inoi [now] should be included into Inoi_base calculation

double holdtime=0
---Time before stress begins (Unit: S)

integer bas_num=6
---number of noise current used to calculate Inoi_base

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
---Compliance of corresponding SMU (Unit: A)

double stress1=3
---Stress value required on the SMU (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
---Compliance of corresponding SMU (Unit: A).

double stress2=3
---Stress value required on the SMU (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
---Compliance of corresponding SMU (Unit: A)

double stress3=nil
---Stress value required on the SMU (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
---Compliance of corresponding SMU (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.
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.

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

Example call

```python
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
```
### NBTI

The NBTI script is used to perform the NBTI test. It supports 2 to 4 SMUs. The Gate performs the STRESS part of the test; the Drain performs the measurement part of the test. In most cases, 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).

**Syntax:**

```plaintext
NBTI(smu_D, smu_G, smu_S, smu_B, vg_stress, vd_stress, vg_meas, vd_meas,
myNPLC, meas_delay, navg, t_array, modeflag, compliancei, time, did)
```

**Inputs:**

- `instid smu_D=SMU1` -- SMU1, SMU2, SMU3, ..., SMU64
- `instid smu_G=SMU2` -- SMU1, SMU2, SMU3, ..., SMU64
- `instid smu_S=KI_GND` -- SMU1, SMU2, SMU3, ..., SMU64, KI_GND
- `instid smu_B=KI_GND` -- SMU1, SMU2, SMU3, ..., SMU64, KI_GND
- `double vg_stress=-2.0 in [-40,40]` -- Gate stress voltage
- `double vd_stress=0 in [-40,40]` -- Drain stress voltage
- `double vg_meas=-1.2 in [-40,40]` -- Gate measure voltage
- `double vd_meas=-1.2 in [-40,40]` -- Drain measure voltage
- `double myNPLC=0.001 in [0.001,25]` -- NPLC, 0.001 ~ 10
- `double meas_delay=0 in [0,]` -- Measure delay after stress is off
- `integer navg=1 in [1,20]` -- Double of points for average
- `table t_array={1,2,5,10,20,50,100}` -- Stress time array
- `integer modeflag=1 in [0,1]` -- Gate first or Drain first
- `double compliancei=0.1 in [0,]` -- Current compliance

**Outputs:**

- `time={} -- 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 29: GUI for NBTI

- Terminal settings: SMUs are assigned to terminals. For Source and Bulk, KI_GND could be set (manual connection). Voltage is changeable only on Gate and Drain. Measurement is made on 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 on which average is taken.
- Gate/Drain checkbox: Voltages applied on Gate and Drain change when measurement begins and ends. The Gate/Drain checkbox is used to determine which terminal will change first. If the Gate Change First checkbox is selected, the Gate terminal changes first. If the Gate Change First checkbox is left deselected, the Drain terminal changes first.
- Stress Array: Used to input the time array.

Example call

```
local compliance=1e-1
local modeflag=0
local vd_meas=0.1
local navg=1
local t_array={0,1,2,5,10,20}
```
local smu_B=SMU4
local smu_D=SMU2
local smu_G=SMU3
local myNPLC=0.01
local vg_meas=1.5
local meas_delay=0
local smu_S=SMU1
local vd_stress=0
local vg_stress=2
local time={}
local did={}
NBTI(smu_D,smu_G,smu_S,smu_B,vg_stress,vd_stress,vg_meas,vd_meas,myNPLC,meas_delay,
    navg,t_array,modeflag,compliancei,time,did)

NBTI_meas

This module performs 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_Vangei,p_sweepdelay,a,b,A,W,L,Vg_ini,Vd_ini,Vg_stress,Vd_stress,Vg_meas,Vd_meas,myNPLC,meas_delay,
    inter_delay,t_mode,t_max,npdec_delta,time_input,modeflag,Gcompi,Dcompi,rng,Nsam)

Inputs:

instid smuD=SMU2 -- SMU1, SMU2, SMU3,..., SMU64
instid smuG=SMU1 -- SMU1, SMU2, SMU3,..., SMU64
instid smuS=KI_GND -- SMU1, SMU2, SMU3,..., SMU64, KI_GND
instid smuB=KI_GND -- SMU1, SMU2, SMU3,..., SMU64, KI_GND

integer flag0=1 in [0,1] meas disable it
integer flag1=1 in [0,1] measure test, "0" means disable it
integer flag2=1 in [0,1] 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 train pulses
integer t_mode=1 in [0,2]        -- "0" for time array given by customer; "1" for logrithmic
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 -- current of drain 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
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 30: GUI for NBTI_meas
Example call

```python
local p_sweepdelay=1e-4
local Vd_ini={0.6,0.7,0.8}
local modeflag=1
local p_Vds=0.5
local Ns a m=5
local npdec_delta=1
local meas_delay=1e-4
local t_max=10
local Gcompi=1e-1
local t_mode=2
local Gcompi=1e-1
local time_input={0,1,2,3,4,5,6,7,8,9,10}
local inter_delay=5e-4
local flag2=1
local flag1=1
local flag0=1
local Vd_stress=0
local myNPLC=0.005
local A=1e-5
local Vg_ini={0.8,0.9,1.0}
local rng={1e-5,1e-4,1e-4}
local L=1
local p_Drangei=0
local p_Vg_hi=2
local W=1
local p_Vg_points=101
local p_Vg_lo=0
local a=0.1
local b=0.1
local Vd_meas={0.6,0.7,0.8}
local smuS=KI_GND
local smuB=KI_GND
local Vg_meas={0.8,0.9,1.0}
local smuG=SMU1
local smuD=SMU2
local Vg_stress=2
NBTI_meas(smuD,smuG,smuS,smuB,flag0,flag1,flag2,p_Vg_lo,p_Vg_hi,p_Vg_points,p_Vds,p_Drangei,p_sweepdelay,a,b,A,W,L,Vg_ini,Vd_ini,Vg_strength,Vd_strength,Vg_meas,Vd_meas,myNPLC,meas_delay,inter_delay,t_mode,t_max,npdec_delta,time_input,modeflag,Gcompi,Gcompi, rng,Ns a m)
```
**NBTI_on_the_fly**

- 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 STRESS phase.
- Vg_relax is for measurement during recovery.
- 0 is set for recovery voltage during time other than measurement.
- This program can only be used for one device; one stress-on period and one 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:

```python
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 31: GUI for NBTI_on_the_fly**

![NBTI on-the-fly GUI](image-url)
- Terminal Connection: For Source and Bulk, KI_GND could be set (manual connection to ground). If specific SMUs are assigned to these two terminals, 0V will be applied internally.

- Gate/Drain voltage setup entry fields: Voltage during stress phase and relax phase on gate and drain should be set here.

- Time setup radio functions: Arranges time during stress and relaxation. For 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, measurement is made during relax phase only following the monitor time array, and stress time is decided by Stress time. If both are selected, measurement is made during both the stress phase and the relax phase, and they both follow monitor time array.

Example call

```lua
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:**

```lua
function qbd_rmpj(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use,
l_init, l_start, F, t_step, exit_volt_mult, V_max, l_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 l_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 l_start=1e-5 in [0.1,0.1] --starting current for current ramp (A). Typical value is 
l_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 l_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
l_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 l_init in pre test
V_init_post --voltage at l_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.
NOTE
If the above routine is modified, change the function name to avoid possible programming errors.

Example call
local HiSMUId=1
local LoSMUId1=2
local LoSMUId2=0
local LoSMUId3=0
local myplc=1
local v_use=0.005
local I_init=1e-8
local I_start=1e-8
local F=1.5
local t_step=0.1
local exit_volt_mult=0.85
local V_max=20
local I_max=1e-5
local q_max=0.1
local area=1
qbd_rmpj(HiSMUId, LoSMUId1, LoSMUId2, LoSMUId3, myplc, v_use, I_init, I_start, F, t_step, exit_volt_mult, V_max, I_max, q_max, area).

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:

\[ \text{qbd_rmpv}(\text{HiSMUId}, \text{LoSMUId1}, \text{LoSMUId2}, \text{LoSMUId3}, \text{myplc}, \text{v_use}, \text{I_init}, \text{hold_time}, \text{v_start}, \text{v_step}, \text{t_step}, \text{measure_delay}, \text{I_crit}, \text{I_box}, \text{I_max}, \text{exit_curr_mult}, \text{exit_slope_mult}, \text{q_max}, \text{t_max}, \text{v_max}, \text{area}, \text{exit_mode}) \]

Inputs:

- \text{integer HiSMUId}=1 \text{ in } [0, 1, 2, \ldots, 64] \quad \text{--maximum 4 smus are supported. if not input '0'}
- \text{integer LoSMUId1}=0 \text{ in } [0, 1, 2, \ldots, 64] \quad \text{--maximum 4 smus are supported. if not input '0'}
- \text{integer LoSMUId2}=0 \text{ in } [0, 1, 2, \ldots, 64] \quad \text{--maximum 4 smus are supported. if not input '0'}
- \text{integer LoSMUId3}=0 \text{ in } [0, 1, 2, \ldots, 64] \quad \text{--maximum 4 smus are supported. if not input '0'}
- \text{double myplc}=1 \text{ in } [0.001, 25] \quad \text{--PLC setting}
- \text{double v_use}=1 \text{ in } [-200, 200] \quad \text{--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.}
- \text{double I_init}=0.001 \text{ in } [-0.1, 0.1] \quad \text{--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.}
- \text{double holdtime}=0 \text{ in } [0, ) \quad \text{--time after V_use is applied (Unit:s)}
- \text{double v_start}=0.01 \text{ in } [-200, 200] \quad \text{--starting ramp voltage (V). Typical value is v_use}
- \text{double v_step}=0.01 \text{ in } [-200, 200] \quad \text{--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.}
- \text{double t_step}=0.1 \text{ in } [0, ) \quad \text{--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.}
- \text{double measure_delay}=0.05 \text{ in } [0, ) \quad \text{--time delay for measurement after each voltage stress step(Unit:s); This delay should be less than t_step.}
- \text{double I_crit}=5e-4 \text{ in } [-0.1, 0.1] \quad \text{--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)}
- \text{double I_box}=3e-4 \text{ in } [-0.1, 0.1] \quad \text{--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.}
- \text{double I_max}=1e-3 \text{ in } [-0.1, 0.1] \quad \text{--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^2)

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^2)

integer exit_mode=0 in(0,1) --failure criteria mode 0:judge by current (I_max) and (exit_curr_mult) and q_max, v_max, t_max 1:also judge slope (exit_slope_mult)

Outputs:

V_stress --voltage stress array
l_stress --measured current array
T_stress --time stamp array representing when current is measured
q_stress --accumulated charge array PER OXIDE AREA
l_use_pre --Measured oxide current at v_use prior to starting the ramp
l_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^2)
v_bd --applied voltage at the step just before oxide breakdown
l_bd --measured current at v_bd just before oxide breakdown
t_bd --time stamp when measuring l_bd
v_crit --applied voltage at the step when the oxide current exceeds l_crit
v_box --applied voltage at the step when the oxide current exceeds l_box
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.
Example call

```python
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 Electrmigration 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/cm²)
- `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 DUT 3, 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 Iso_EM test.
Figure 34: GUI for ISO_EM test

**Example call**

```python
local n_narrow=1
local T_target=50
local test_mode=0
local I_limit=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, I_limit, init_failR, Tinit, start_J, step_J, step_delay, equil_time, T_target, 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 saved in this path: \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 ACS Reference manual (document number: ACS-901-01).

Configure a capacitor meter library

CV_4200CVU and KI42xxCVU

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

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

Figure 35: Select 4200CVU user module
The details of the Model 4200-CVU GUI are indicated in the graphic and with a description that follows (see next Figure).

**Figure 36: CV_4200CVU setting example**

![Image of the GUI](image)

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** (double): Hold time after force value changed. Unit: second.
- **delay_time** (double): Delay before each measurement (0 to 999s). Unit: second.
- **Speed** (int): KI.CVU_SPEED_FAST = 0; KI.CVU_SPEED_NORMAL = 1; KI.CVU_SPEED_QUIET = 2

**Force Func Parameters**:
- **preSoak**: Force voltage after test start and before measurement sequence (units are in volts)

**Bias Setup** (enabled by clicking Bias in Force Func)
- **Bias**: force value for the bias (units are in volts)
- **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 (e.g., 15kHz input will click 20kHz). Unit: Hz.
- Voltage: Voltage level of the AC drive (10mV to 100mVRMS). Unit: mVRMS

Measure Setting

- measParam: 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

- measRange: Current measure range for impedance measurements. Setting range to zero clicks auto range.

Compensation Setting

- cableLength (float): Cable length setting for connect compensation. unit: meter. Values from zero to 3 are valid, but only 0, 1.5, and 3 are supported lengths. Any other number from zero to 3 will be changed to one of the three values. When you do not need compensation, cable length should be assigned to zero.
- Connection Compensation
- Open: enable or disable compensation constants for open
- Short: enable or disable compensation constants for Short
- Load: enable or disable compensation constants for Load output parameter name setting: input wanted name for output parameter

The GUI output error list:

- 0: OK
- -10000: Specified CVU does not exist
- -10001: (INVAL_PARAM) Parameter setting error occurred
- -10090: (GPIB_ERROR_OCCURED) 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, freqbias, v_AC, meas_param, meas_range, cable_length, isCmpstOpen, isCmpstShort, isCmpstLoad, output_DCv, output_result1, output_result2, output_error)
```

**CV_HP4284**

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

Instrument: Agilent 4284 or 4980 LCR meter

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

Figure 37: Select HP4284 user module
The GUI inputs are as follows:

CMTR_name (string): Instrument name defined in syscon.kcf file

force_func (int): 0: bias, 1: sweep

preSoak (double): Force voltage after test start and before measurement sequence

v_bias (double): Force value for the bias

v_biasPts (int): The number of bias points

v_start (double): Initial force value for the sweep (-40V to 40V)

v_stop (double): Final force value for the sweep (-40V to 40V)

v_step (double): Step force value for the sweep (-40V to 40V)

hold_time (double): Hold time after force value changed

delay_time (double): Delay before each measurement (0 to 999s)

Speed (string): Measurement time setting

freq_bias (double): Frequency of the AC drive for normal measurement. Valid value is from 20 Hz to 2MHz

vAC (double): The oscillator output voltage level, the valid input is 5mV to 20V
measParam: 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"

measRange: The measurement range to use. Valid values for this parameter are 0 (Auto), 100, 300, 1000, 3000, 10000, 30000, and 100000 Ohms

cableLength (float): Cable length setting for connect compensation. Unit: meter. When you do not need compensation, cable length should be assigned to 0

isCmpstOpen (bool): enable or disable compensation constants for open

isCmpstShort (bool): enable or disable compensation constants for Short

isCmpstLoad (bool): enable or disable compensation constants for Load

The GUI output error list:

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-10000</td>
<td>Specified CVU does not exist</td>
</tr>
<tr>
<td>-10001</td>
<td>(INVAL_PARAM) Parameter setting error occurred</td>
</tr>
<tr>
<td>-10090</td>
<td>(GPIB_ERROR_OCCURED) A GPIB communications error occurred</td>
</tr>
</tbody>
</table>

return dictionary:

result["DCV"]: Force DC voltage

result["result1"]: The first parameter of the result according to the measurement model.

result["result2"]: The second parameter of the result according to the measurement model.

Syntax:

CVITM.cv4284(CMTR_name, force_func, preSoak, v_bias, v_biasPts, v_start, v_stop, v_step, hold_time, delay_time, speed, freq_bias, v_AC, meas_param, meas_range, cable_length, isCmpstOpen, isCmpstShort, isCmpstLoad, output_DCV, output_result1, output_result2, output_error)
Configure a switch matrix library

**Switch_Control**

Function: Add a PTM to test tree, then from PTMLib import switchctrl.py module. The Switch GUI will display. (see next Figure).

**Figure 40: 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:

- Click the switch matrix from the Switch Name sub-list box.
- Click the cells on the panel and the related rows and columns of the matrix will connect. For example, click the A1, 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 canceled.
- The Clear All function will clear all setup connections.
- If the Open All is selected, the matrix will open all old connections before connecting.

Figure 41: Switch_Control setting example
**Series 3700 System Switch**

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

Instrument: Keithley Instruments Series 3700 System Switch/Multimeter Cards

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

**Figure 42: Series 3700 System 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 by clicking the Matrix Card.
3. Click the slot number from 1 to 6.
4. Click the cells on the panel, 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 canceled.
5. If you want to clear all connections, click the Open All function.
To control the multiplexer card from the GUI:

1. Input the GPIB address number in the GPIB edit box.
2. Click the card tab by clicking the multiplexer card.
3. Click the slot number from 1 to 6.
4. Click the cells on the panel, the related rows and columns of 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 canceled.
If want to accomplish a DMM function test, select the DMM Function, then set the range (0 means auto-range), nplc and click test function (see next Figure).

Figure 46: 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
......

$1Channel16

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

Add a PTM to the test tree, then from the PTMLib import the TEKSCOPE.py module. The TEKSCOPE GUI will display (see next Figure).

Figure 47: TEKSCOPE read wave test module GUI
**TEKSCOPE_ReadWave**

Function: Read the waveform from the scope. This module reads data from one channel at a time. Some modifications are needed to enable it to read data from more channels simultaneously (see next Figure).

*Known issues:* There is a known issue when returning 2 bytes of binary data.

Instrument: TEKSCOPE

**Figure 48: Waveform reading data**

Configure a Series 23x library

Add a PTM to the test tree, then from the PTMLib import the ki237Test.py module. The 237 test library GUI will display. Click the desired test module from the Test Module sublist box.

**BiasVolt_SampleCurr**

Function: 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 Units

*Inputs:*

- **instAddr:** GPIB address, 0 through 30, default is 17, change the address according to instrument setting.

- **biasV:** Bias Voltage. Limit of value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.
RangeV: Source range: 0 is auto-range, otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by different Models of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

DelayV: Source Delay, 0 through 65000, default 0.

Compliance: Current compliance of the sweep. 1E-9 through 1E-1. Value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

RangeCurr: Measurement range for current. 0 through 9. 0 is auto-range, otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

NumSamples:

interval: Sampling interval,(s), Valid input 0 to 1000 s.

Outputs:

output_Curr: Measured current

output_time: Timestamp at each point

output_error: Error value

0: OK

-10090: GPIB_ERROR_OCCUR
Sweepsystem computer_23x

Function: This module is used to sweep current and take I/V/Time readings for 236/237/238.

Instrument: Keithley Instruments Model 236/237/238 Source Measure Units

Inputs:

instAddr: GPIB address, 0 through 30, default is 17; change the address according to instrument setting.

SweepMode: Sweep mode. 0: fixed bias, sampling measurement 1: Linear sweep; 2: Log sweep.

StartI: Start current of the sweep. If in sampling mode, this is the output source value. Limit of value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

StopI: Stop current of the sweep. Limit of value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

NumOfPoints: Number of sweep points. Valid input 1 to 1000 for fixed bias mode and linear mode. For log sweep mode, valid input is:

0: 5 points per decade sweep
1: 10 points per decade sweep
2: 25 points per decade sweep  
3: 50 points per decade sweep  

ComplianceV: Voltage compliance of the sweep. Value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

SourceRange: Source range. 0: Auto-range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

MeasureRange: Measurement range for current. 0: Auto-range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

HoldTime: Time sitting at the first point of sweep. Valid input 0s to 9999.999 seconds.

SweepDelay: Delay time between each sweep point. Valid input 0s to 9999.999 seconds.

Integration: A/D integration speed:  
0: fast  
1: medium  
2: long, 1PLC (60Hz)  
3: long, 1PLC (50Hz)

Outputs:  
output_V: Measured voltage  
output_I: Measured current  
output_time: Timestamp at each point  
output_error: Error value  
0: OK  
-10090: GPIB_ERROR_OCCUR  
-10100: INVAL_PARAM
Figure 50: 23x Sweep standard GUI

**SweepVolt_23x**

Function: This module is used to sweep voltage and take I/V/Time readings for 236/237/238. Run config23x prior to this module.

Instrument: Keithley Instruments Model 236/237/238 Source Measure Units

**Inputs:**

- **instAddr:** GPIB address, 0 through 30, default is 17; change the address according to instrument setting.

- **SweepMode:** Sweep mode. 0: fixed bias sampling measurement. 1: Linear sweep; 2: Log sweep.

- **StartV:** Start Voltage of the sweep. If in sampling mode, this is the output bias value. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

- **StopV:** Stop Voltage of the sweep. Limit of value differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

- **NumofPoints:** Number of sweep points. Valid input 1 to 1000 for fixed bias mode and linear mode. For log sweep mode, valid input is:
  
  - 0: 5 points per decade sweep
  - 1: 10 points per decade sweep
2: 25 points per decade sweep
3: 50 points per decade sweep

Compliance: Current compliance of the sweep. Limit of value differs by different Model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

Source Range: Source range. 0: Auto-range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

Measure Range: Measurement range for current. 0: Auto-range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

Hold Time: Time sitting at the first point of sweep. Valid input 0s to 9999.999 seconds.

Sweep Delay: Delay time between each sweep point. Valid input 0s to 9999.999 seconds.

Integration: A/D integration speed:
  0: fast
  1: medium
  2: long, 1PLC (60Hz)
  3: long, 1PLC (50Hz)

Outputs:

output_V: Measured voltage
output_I: Measured current
output_time: Timestamp at each point
output_error: Error value

  0   OK.
  -1  23x not found on GPIB
  -10000 (INVAL_INST_ID) The specified instrument ID does not exist.
  -10090 (GPIB_ERROR_OCCURRED) A GPIB communications error occurred.
  -10091 (GPIB_TIMEOUT) A timeout occurred during communications.
  -10100 (Invalid Parameter) An error occurred on an input parameter.
VdIs_237

Function: HV Measurement of current Id while forcing Vd with stepping Vg.

Instrument: Keithley Instruments Model 236/237/238 Source Measure Units, Model 4200
Semiconductor Characterization System.

Device Connection:

Drain: KI237

Gate, Source, Sub, Well: Each corresponds with one of SMU1 or SMU2 or .... or SMU8 of 4200.

Inputs:

instAddr: GPIB address of 237. Valid from 0 through 30.

GateSMU: The system terminal connected to the Gate of the MOS-FET. If 'GNDU' is chosen, the terminal should be connected to GND manually.

SourceSMU: The system terminal connected to the Source of the MOS-FET. If 'GNDU' is chosen, the terminal should be connected to GND manually.

SubSMU: The system terminal connected to the Sub of the MOS-FET. If 'GNDU' is chosen, the terminal should be connected to GND manually.

WellSMU: The system terminal connected to the Well of the MOS-FET. If 'GNDU' is chosen, the terminal should be connected to GND manually; if there is not a Well terminal, choose 'NONE'.
**VgStart:** Start Voltages of the Gate in volt.

**VgStop:** End Voltages of the Gate in volt.

**VgPoint:** Number of intervals of forced Vg.

**VdStart:** Start Voltages of the Drain in volt.

**VdStop:** End Voltages of the Drain in volt.

**VdPoint:** Number of intervals of forced Vd.

**IdLimit:** Current Limit on measured sites in Ampere.

**Integration:** A/D integration speed:

0: fast
1: medium
2: long, 1PLC (60Hz)
3: long, 1PLC (50Hz)

**DelayTime:** Delay Time of one measurement in seconds.

**VscForce:** Voltage bias force to Source.

**VsbForce:** Voltage bias force to Sub.

**VwForce:** Voltage bias force to Well.

**VgMsrFlag:** Flag for determining if Vg is measured.

**IgMsrFlag:** Flag for determining if Ig is measured.

**VscMsrFlag:** Flag for determining if Vsc is measured.

**IscMsrFlag:** Flag for determining if Isc is measured.

**VsbMsrFlag:** Flag for determining if Vsb is measured.

**IsbMsrFlag:** Flag for determining if Isb is measured.

**VwMsrFlag:** Flag for determining if Vw is measured.

**IwMsrFlag:** Flag for determining if Iw 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 52: Model 237 VdsId standard GUI

![Model 237 VdsId standard GUI](image1)

Figure 53: Model 237 VdsId test result

![Model 237 VdsId test result](image2)
Configure a Series 3700 system switch DMM library

**Series 3700 System Switch DMM 2-Wire**

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

**Figure 54: 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 DMM.

Pin Connection: If 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 4-Wire**

Add a PTM to the test tree, then from the PTMLib import the FourWireResistor.py module. The 3700 DMM resistor test GUI will display (see next Figure).
Figure 55: 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 DMM.

Pin Connection: If 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, Input LO terminal to Pin (2 or 9), Sense HI to Pin3, and Sense LO to Pin4.

Configure a Series 2400 SourceMeter instruments library

Add a PTM to the test tree, then from the PTMLib import the HiPower_24.py module. The 2400 test library GUI will display. Click the wanted test module from Test Module sub-list box.

**Series 2400 SourceMeter instruments_IdVg**

Instrument: Keithley Instruments Model 2400 SourceMeter®

Function: This module is used to test drain current at a specified Drain voltage and Gate voltage sweep.

Pin Connection: Gate swept, Drain biased. Bulk and Source connected to ground if there is no applied voltage.

Results:

- Get measured Drain current at Gate voltage sweep.
• Get results Vtx and Vt0.

**Input:**

- **drain_addr** (int): Drain terminal 2430 GPIB address.
- **gate_addr** (int): Gate terminal Series 2400 SourceMeter instruments GPIB address.
- **vg_start** (double): Start voltage of Gate.
- **vg_stop** (double): Stop voltage of Gate.
- **points** (int): The number of points for Gate sweep.
- **vd** (double): Drain bias voltage.
- **hold_time** (double): Hold time in second before Gate sweep. The valid value is from 0 to 9999.999.
- **delay_time** (double): Delay time in seconds between each Gate sweep point. The valid value is from 0 to 9999.999.
- **limiti** (double): Compliance value for drain voltage force. The valid input is from -10A to 10A.
- **rangei** (double): The current range for drain current measure. For pulse mode, auto range is not allowed. Valid input: -10 through 10.
- **plc** (double): Number of Power Line Cycles for integration time. For pulse mode, valid input 0.004 to 0.1.

**Output:**

- **Id** (D_ARRAY_T) Drain current measured at Gate sweep voltage
- **Vg** (D_ARRAY_T) Gate voltage programmed
- **Gm** (D_ARRAY_T) Gm=dId/dVg
- **Vtx** (double*) Vtx = Vt0-Vs/2

**Error:** Error value

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>-1</td>
<td>Series 2400 SourceMeter instruments not found on GPIB.</td>
</tr>
<tr>
<td>-200</td>
<td>Initialize error occurred.</td>
</tr>
<tr>
<td>-10000</td>
<td>(INVAL_INST_ID) The specified instrument ID does not exist.</td>
</tr>
<tr>
<td>-10100</td>
<td>(INVAL_PARAM) Parameter setting error occurred.</td>
</tr>
<tr>
<td>-10090</td>
<td>(GPIB_ERROR_OCCURRED) A GPIB communications error occurred.</td>
</tr>
<tr>
<td>-10091</td>
<td>(GPIB_TIMEOUT) A timeout occurred during communications.</td>
</tr>
</tbody>
</table>
Series 2400 SourceMeter instruments_IdVg_Pulse

Module Type: PTM

Instrument: Keithley Instruments Model 2400 SourceMeter®, at least one Model 2430 SourceMeter®

Function: This module is used to test drain current during a Gate voltage sweep at a specified Drain voltage, with measurement at the Drain terminal in pulse mode using the Model 2430 SourceMeter controlled over the GPIB bus only.

Pin Connection: Gate sweep, Drain bias. Bulk and Source connected to ground if there is no applied voltage.

Results:
• Get measured Drain current during the Gate voltage sweep with the Drain in pulse mode.
• Get results Vtx and Vt0.

**Input:**

```
drain_addr (int): Drain terminal 2430 GPIB address.
gate_addr (int): Gate terminal Series 2400 S ourceMeter instruments GPIB address.
vg_start (double): Start voltage of Gate. The valid input is from -200V to 200V.
vg_stop (double): Stop voltage of Gate. The valid input is from -200V to 200V.
points (int): The number of points for Gate sweep.
vd (double): Drain bias voltage.
hold_time (double): Hold time in second before Gate sweep. The valid value is from 0 to 9999.999.
delay_time (double): Delay time in second between each Gate sweep point. The valid value is from 0 to 9999.999.
limiti (double): Compliance value for drain voltage force. The valid input is from -10A to 10A.
rangeni (double): The current range for drain current measure. For pulse mode, auto range is not allowed. Valid input: -10 through 10.
plc (double): Number of Power Line Cycles for integration time. For pulse mode, valid input 0.004 to 0.1.
pulse_width (double): Output in time of the pulse for drain voltage force. The valid input is from 150us 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): Output off duration for the pulse of the drain voltage force. The valid input is from 0 to 9999.999.
```

**Output:**

```
Id (D_ARRAY_T) Drain current measured at Gate sweep voltage.
Vg (D_ARRAY_T) Gate voltage programmed
Gm (D_ARRAY_T) Gm=dId/dVg
Vtx (double*) Vtx= Vt0-Vs/2
```
Vt0 (double*) Calculate Gm=dl/dVg. Find Gmmax and extrapolate back to Ids=0 to find Vt0

Error: Error value

0  OK.
-1  Series 2400 SourceMeter instruments not found on GPIB.
-2  2430 not found on GPIB.
-200  Initialize error occurred.
-300  Configuration error occurred.
-400  Reading error occurred.
-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 58: Model 2430 IdVg Pulse
**Series 2400 SourceMeter instruments_IdVd**

Module Type: PTM

Instrument: Keithley Instruments Model 2400 SourceMeter®

DUT: MOSFET, source and bulk to be grounded

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

Pin Connection: Drain swept, Gate biased. Bulk and Source connected to ground if there is no applied voltage.

Results: Get measured Drain current during the Drain voltage sweep at 10 Gate bias voltages.

*Input:*

`drain_addr` (int): Drain terminal Series 2400 SourceMeter instruments GPIB address.

`gate_addr` (int): Date terminal Series 2400 SourceMeter instruments GPIB address.

`vd_start` (double): Start sweep pulse voltage of Drain.

`vd_stop` (double): Stop sweep pulse voltage of Drain.

`points` (int): The number of points for Drain sweep.

`limit` (double): Compliance value for drain voltage force. The valid input is from -10A to 10A.

`range` (double): The current range for drain current measure. For pulse mode, auto range is not allowed. Valid input: -10 through 10.

`plc` (double): Number of Power Line Cycles for integration time. For pulse mode, valid input 0.004 to 0.1.

`vg_start` (double): Start voltage of gate.
vg_stop (double): Stop voltage of gate.

vg_step (double): Step voltage of gate.

hold_time (double): Sweep first point hold time.

delay_time (double): Sweep delay time.

Output:

Vd (D_ARRAY_T) Drain voltage programmed.

Id1 (D_ARRAY_T) Drain current measured at the 1st Gate bias voltage.

Id2 (D_ARRAY_T) Drain current measured at the 2nd Gate bias voltage.

Id3 (D_ARRAY_T) Drain current measured at the 3rd Gate bias voltage.

Id4 (D_ARRAY_T) Drain current measured at the 4th Gate bias voltage.

...

Error: Error value

0 OK.

-1 Series 2400 SourceMeter instruments not found on GPIB.

-200 Initialize error occurred.

-400 Reading error occurred.

-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 60: Series 2400 instruments IdVd**

**Figure 61: Series 2400 instruments IdVd test results**

**Series 2400 SourceMeter instruments IdVd_Pulse**

**Module Type:** PTM

**Instrument:** Keithley Instruments Model 2400 SourceMeter®, at least one Model 2430 SourceMeter®

**Function:** This module is used to test drain current at a specified Gate voltage during a Drain voltage sweep, with measurement at the Drain terminal in sweep pulse mode using the Model 2430 SourceMeter controlled over the GPIB bus only.
**Input:**

- **drain_addr** (int): drain terminal 2430 GPIB address.
- **gate_addr** (int): gate terminal Series 2400 SourceMeter instruments GPIB address.
- **vd_start** (double): Start pulse voltage of Drain.
- **vd_stop** (double): Stop pulse voltage of Drain.
- **points** (int): The number of points for Drain sweep.
- **limiti** (double): Compliance value for drain voltage force. The valid input is from -10A to 10A.
- **rangei** (double): The current range for drain current measure. For pulse mode, auto range is not allowed. Valid input: -10 through 10.
- **plc** (double): Number of Power Line Cycles for integration time. For pulse mode, valid input 0.004 to 0.1.
- **vg_start** (double): Start voltage of gate.
- **vg_stop** (double): Stop voltage of gate.
- **vg_step** (double): Step voltage of gate.
- **pulse_width** (double): Duration of the output ON time. The valid value is from 0.15ms to 5ms.
- **pulst_delay** (double): Duration of the output OFF time. The valid value is from 0s to 9999.999s.

**Output:**

- **Vd** (D_ARRAY_T) Drain voltage programmed.
- **Idi** (D_ARRAY_T) Drain current measured at the first Gate bias voltage.

**Error:**

- 0  OK.
- -1  Series 2400 SourceMeter instruments not found on GPIB.
- -2  2430 not found on GPIB.
- -200  Initialize error occurred.
- -300  Configuration error occurred.
- -400  Reading error occurred.
- -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.
Series 2400 SourceMeter instruments_BiasV_Pulse

Module Type: PTM

Instrument: Keithley Instruments Model 2430 SourceMeter®

Function: This module is used to perform Pulse V measure I with Model 2430 in pulse mode. SourceMeter controlled over the GPIB bus only.

Results: Get results by forcing V pulse.
Input:

- **gpib_addr** (int): Instrument GPIB address. Valid input: 0 through 30.
- **voltage** (double): Pulse level forced.
- **points** (int): The number of forced pulses. Valid input: 1 through 2500.
- **pulse_width** (double): Duration of the output ON time. Unit: second. Valid value: 0.15ms to 5ms.
- **pulse_delay** (double): Duration of the output OFF time. Unit: second. Valid value: 0s to 9999.999s.
- **plc** (double): Number of Power Line Cycles for integration time. In pulse mode, valid input: 0.004 to 0.1.
- **limiti** (double): Compliance value.
- **rangei** (double): The range for measurement.

Output:

Error: Error value

- 0: OK.
- -1: Series 2400 SourceMeter instruments not found on GPIB.
- -2: 2430 not found on GPIB.
- -200: Initialize error occurred.
- -300: Configuration error occurred.
- -400: Reading error occurred.
- -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.
Module name: BiasIMeasV_Pulse

Instrument: Keithley Instruments Model 2430 SourceMeter®.

Function: This module is used to perform Pulse I measure V with Model 2430 in pulse mode. SourceMeter controlled over the GPIB bus only.

Results: Get results by forcing I pulse.

Inputs:


`current` (double): Pulse level forced.

`points` (int): The number of forced pulses. The valid input: 1 through 2500.

`pulse_width` (double): Duration of the output ON time. Unit: second. The valid value is from 0.15ms to 5ms.

`pulse_delay` (double): Duration of the output OFF time. Unit: second. The valid value is from 0s to 9999.999s.
plc (double): Number of Power Line Cycles for integration time. In pulse mode, valid input 0.004 to 0.1.

limitv (double): Compliance value.

rangev (double): The range for measurement.

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 Initialize error occurred.
-300 Configuration error occurred.
-400 Reading error occurred.
-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 65: Series 2400 instruments Bias pulse standard GUI
Series 2400 SourceMeter instruments_SweepV

Module name: SweepV_MeasI


Function: This module is used to sweep V signal and take I/V/Time readings for 2400/2410/2420/2425/2430.

Inputs:


startv (double): Start signal level of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

stopv (double): Stop signal level of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

points (int): Number of sweep points. Valid input 1 to 2500.

sweepMode (int): Sweep mode. 0: Linear sweep; 1: Log sweep.

limiti (double): Compliance of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

srangev (double): Source range. 0: Auto range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

mrangei (double): Measurement range for current. 0: Auto range. Otherwise, the range is the smallest that can accommodate the input value. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

hold_time (double): Time sitting at the first point of sweep. Valid input: 0s to 9999.999 seconds.

delay_time (double): Delay time between each sweep point. Valid input: 0s to 9999.999 seconds.

plc (double): A/D integration time in term of Power Line Cycles (PLCs). Valid input 0.01 to 10.

Outputs:

I

V

time

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

Figure 66: Series 2400 instruments SweepV standard GUI

Series 2400 SourceMeter instruments_SweepI

Module name: SweepI_MeasV


Function: This module is used to sweep I signal and take I/V/Time readings for 2400/2410/2420/2425/2430.

Inputs:


starti (double): Start signal level of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.
stopi (double): Stop signal level of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

points (int): Number of sweep points. Valid input 1 to 2500.

limitv (double): Compliance of the sweep. Limit values differ by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

srangei (double): Source range. 0: Auto range. Otherwise, the range is the smallest that can accommodate the input value. The source range limit differs by model of the meter. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

mrangev (double): Measurement range for voltage. 0: Auto range. Otherwise, the range is the smallest that can accommodate the input value. Refer to the ACS Reference manual (document number: ACS-901-01) for more information.

hold_time (double): Time sitting at the first point of sweep. Valid input: 0s to 9999.999 seconds.

delay_time (double): Delay time between each sweep point. Valid input 0s to 9999.999 seconds.

plc (double): A/D integration time in term of Power Line Cycles (PLCs). Valid input 0.01 to 10.

Outputs:

I
V
time

Error: Error value

0 OK.
-200 Instrument initialize error.
-300 Configuration error occurred.
-400 Reading error occurred.
-10000 (INVAL_INST_ID) The specified instrument ID does not exist.
-10090 (GPIB_ERROR_OCCURRED) A GPIB communications error occurred.
-10091 (GPIB_TIMEOUT) A timeout occurred during communications.
Figure 67: Series 2400 instruments SweepI standard GUI

![Figure 67: Series 2400 instruments SweepI standard GUI](image_url)
Python Test Module (PTM) Debug Tool

In this section:

- PTM debug tool introduction .................................................... 4-1
- PythonWin description ............................................................. 4-1
- PTM debugging ........................................................................ 4-4
- Debug tool limitations ............................................................. 4-19

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 description

PythonWin Debugger is a graphical user interface (GUI) and includes an easy to use interactive editing environment (see next Figure).
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 69: 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 70: 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 71: PythonWin Run Script**

1. Check (checks the current file without executing it)
2. Interactive window (show or hide the interactive window)
PTM debugging

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 72: 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 73: 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.

Figure 74: Select PTM for debugging

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).
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 76: Start Debug Run feature

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 77: 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.
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 79: 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 80: 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 81: 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 82: Watch window variables**

![Watch window variables](image)

- **NOTE**

  In the next Figure, note that the Expression (the variable) named "getattr" is 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).
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 84: Create a variable in the Watch window

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vtou</td>
<td>NameError: name &quot;Vtou&quot; is not defined</td>
</tr>
<tr>
<td>data_val</td>
<td>NameError: name 'data_val' is not defined</td>
</tr>
<tr>
<td>data_name</td>
<td>NameError: name 'data_name' is not defined</td>
</tr>
<tr>
<td>Gdata_file</td>
<td>NameError: name 'Gdata_file' is not defined</td>
</tr>
<tr>
<td>ShowMessage</td>
<td>NameError: name 'ShowMessage' is not defined</td>
</tr>
<tr>
<td>getatr</td>
<td>&lt;built-in function getatr&gt;</td>
</tr>
</tbody>
</table>
View the Stack view window

Click the Stack view icon (see next Figure).

Figure 85: 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 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 86: Stack view window

View the Breakpoint list and Toggle Breakpoints

Click the Breakpoint list icon (see next Figure).

Figure 87: 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 88: Breakpoint list window**

<table>
<thead>
<tr>
<th>Condition</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED is good</td>
<td>led_results.py: 25</td>
</tr>
<tr>
<td>Not</td>
<td>object.py: 13</td>
</tr>
<tr>
<td>Not</td>
<td>Edit item</td>
</tr>
<tr>
<td>Not</td>
<td>Delete item</td>
</tr>
<tr>
<td></td>
<td>ct.py: 16</td>
</tr>
<tr>
<td></td>
<td>results.py: 14</td>
</tr>
</tbody>
</table>

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 89: 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 90: 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 91: 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 92: 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 93: 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 94: The Step out icon**

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

**Figure 95: 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 96: The Close icon**
Pythonwin Standard Toolbar icons and functions

Figure 97: 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 98: 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 99: 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 100: 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 101: 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 102: 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 103: Interactive Window

![Interactive Window](image)

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.
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 PTM (Python Test Module), CTM (C language Test Module) or STM (Script Test Module) 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 results data files.

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 (Python Test Modules).

Global variables

Global data variables which can be accessed from a UAP PTM are shown in the table below. All global variables start with the characters "ACS_".
### Table 1: ACS Global Variable

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Type</th>
<th>Comment</th>
<th>UAP Level to Get Value</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACS_frame</td>
<td>string</td>
<td>The frame of ACS</td>
<td>Any UAP</td>
<td>Object</td>
</tr>
<tr>
<td>ACS_uap_level</td>
<td>string</td>
<td>Current UAP level</td>
<td>Any UAP</td>
<td>'wafer_begin', site_end</td>
</tr>
<tr>
<td>ACS_proj_path</td>
<td>string</td>
<td>Current path</td>
<td>Any UAP</td>
<td>'C:\ACS\Projects\default '</td>
</tr>
<tr>
<td>ACS_proj_name</td>
<td>string</td>
<td>Current project name</td>
<td>Any UAP</td>
<td>'default '</td>
</tr>
<tr>
<td>ACS_lot_id</td>
<td>string</td>
<td>Current lot ID</td>
<td>Any UAP except lot begin</td>
<td>'lotid001'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>&quot; (if single wafer)</td>
</tr>
<tr>
<td>ACS_slot_no</td>
<td>int</td>
<td>Current slot number</td>
<td>Wafer level and below</td>
<td>1, 2, …25</td>
</tr>
<tr>
<td>ACS_wafer_id</td>
<td>string</td>
<td>Current wafer ID</td>
<td>Wafer level and below</td>
<td>'waferid001'</td>
</tr>
<tr>
<td>ACS_pattern_id</td>
<td>string</td>
<td>Current pattern ID</td>
<td>Pattern level and below</td>
<td>'Pattern1'</td>
</tr>
<tr>
<td>ACS_site_id</td>
<td>string</td>
<td>Current die ID</td>
<td>Site level and below</td>
<td>'Site_n1p1'</td>
</tr>
<tr>
<td>ACS_site_coord</td>
<td>tuple</td>
<td>Current die coordinates</td>
<td>Site level and below</td>
<td>(-1, 1)</td>
</tr>
<tr>
<td>ACS_ssite_id</td>
<td>string</td>
<td>Current subsite ID</td>
<td>Subsite level and below</td>
<td>'subsite1'</td>
</tr>
<tr>
<td>ACS_ssite_loop</td>
<td>int</td>
<td>Current subsite index of subsite loop</td>
<td>Subsite level and below</td>
<td>1</td>
</tr>
<tr>
<td>ACS_ssite_loopNum</td>
<td>int</td>
<td>Total loop number of the current subsite</td>
<td>Subsite level and below</td>
<td>1</td>
</tr>
<tr>
<td>ACS_ssite_coord</td>
<td>tuple</td>
<td>Current subsite coordinates</td>
<td>Subsite level and below</td>
<td>(0.0,0.0,0)</td>
</tr>
<tr>
<td>ACS_dut_id</td>
<td>string</td>
<td>Current DUT ID</td>
<td>DUT level and below</td>
<td>'Resistor'</td>
</tr>
<tr>
<td>ACS_test_id</td>
<td>string</td>
<td>Current test ID</td>
<td>Test level</td>
<td>'sweepv'</td>
</tr>
<tr>
<td>ACS_test_data</td>
<td>dictionary</td>
<td>Current test data</td>
<td>Test end level</td>
<td>See detail below</td>
</tr>
<tr>
<td>ACS_wdf_head</td>
<td>dictionary</td>
<td>WDF header information</td>
<td>Wafer level and below</td>
<td>See detail below</td>
</tr>
<tr>
<td>ACS_wdf_nonexist</td>
<td>dictionary</td>
<td>WDF erased sites dictionary</td>
<td>Any UAP</td>
<td>See detail below</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Type</td>
<td>Comment</td>
<td>UAP Level to Get Value</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------</td>
<td>----------------------------------------------</td>
<td>------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>ACS_frame</td>
<td>string</td>
<td>The frame of ACS</td>
<td>Any UAP</td>
<td>Object</td>
</tr>
<tr>
<td>ACS_uap_level</td>
<td>string</td>
<td>Current UAP level</td>
<td>Any UAP</td>
<td>'wafer_begin', site_end</td>
</tr>
<tr>
<td>ACS_t_start</td>
<td>string</td>
<td>Test start time</td>
<td>Any UAP</td>
<td>'08-Aug-2008 20:08'</td>
</tr>
<tr>
<td>ACS_t_end</td>
<td>string</td>
<td>Test end time</td>
<td>Any UAP</td>
<td>'08-Aug-2008 20:10'</td>
</tr>
<tr>
<td>ACS_first_wafer</td>
<td>bool</td>
<td>Is this the first wafer in the cassette?</td>
<td>Wafer level and below</td>
<td>True or False</td>
</tr>
<tr>
<td>ACS_last_wafer</td>
<td>bool</td>
<td>Is this the last wafer in the cassette?</td>
<td>Wafer level and below</td>
<td>True or False</td>
</tr>
<tr>
<td>ACS_first_site</td>
<td>bool</td>
<td>Is this the first site of the wafer?</td>
<td>Site level and below</td>
<td>True or False</td>
</tr>
<tr>
<td>ACS_last_site</td>
<td>bool</td>
<td>Is this the last site of the wafer?</td>
<td>Site level and below</td>
<td>True or False</td>
</tr>
<tr>
<td>ACS_output_list</td>
<td>list</td>
<td>All tests' output list</td>
<td>Any UAP</td>
<td>See detail below</td>
</tr>
<tr>
<td>ACS_temp_klf</td>
<td>string</td>
<td>Temporary klf file location</td>
<td>Any UAP</td>
<td>'C:\DOCUMENTS\LOCAL\Temp'</td>
</tr>
<tr>
<td>ACS_prb_errcode</td>
<td>int</td>
<td>Prober error code</td>
<td>Any UAP</td>
<td>0, -1013</td>
</tr>
<tr>
<td>ACS_operator</td>
<td>string</td>
<td>Operator field from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_die_type</td>
<td>string</td>
<td>Die type from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_remark</td>
<td>string</td>
<td>Remark from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_session_name</td>
<td>string</td>
<td>Session name from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_equipment_id</td>
<td>string</td>
<td>Equipment from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_fixture_id</td>
<td>string</td>
<td>Fixture ID from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_testplan_ver</td>
<td>string</td>
<td>Test Plan Version from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_process_level</td>
<td>string</td>
<td>Test Process Level from automation panel</td>
<td>Any UAP, from Automation</td>
<td>Same as entry</td>
</tr>
<tr>
<td>ACS_userdata_path</td>
<td>string</td>
<td>Full path of user data</td>
<td>Any UAP</td>
<td>'C:\ACS\user_data'</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Type</td>
<td>Comment</td>
<td>UAP Level to Get Value</td>
<td>Example</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
<td>----------------------------------------------</td>
<td>------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>ACS_frame</td>
<td>string</td>
<td>The frame of ACS</td>
<td>Any UAP</td>
<td>Object</td>
</tr>
<tr>
<td>ACS_uap_level</td>
<td>string</td>
<td>Current UAP level</td>
<td>Any UAP</td>
<td>'wafer_begin', site_end</td>
</tr>
<tr>
<td>ACS_ptm_path</td>
<td>string</td>
<td>PTM directory setting in preference</td>
<td>Any UAP</td>
<td>'C:\ACS\library\pyLibrary\PTMLib'</td>
</tr>
<tr>
<td>ACS_site_passfail_info</td>
<td>bool</td>
<td>Did the current site pass or fail?</td>
<td>Any UAP</td>
<td>True or False</td>
</tr>
<tr>
<td>ACS_test_wafer_ids</td>
<td>list</td>
<td>Save the IDs of the tested wafers,</td>
<td>Any UAP, from Automation</td>
<td>['01', '02', '03', '04']</td>
</tr>
<tr>
<td>ACS_rpt_each_wafer</td>
<td>int</td>
<td>Create a kdf file for each wafer? (1= yes, 0 = no)</td>
<td>Any UAP</td>
<td>0</td>
</tr>
<tr>
<td>ACS_ktxe_exit</td>
<td>int</td>
<td>Stop the execution of ktxe</td>
<td>Any UAP</td>
<td>1</td>
</tr>
<tr>
<td>ACS_ktxe_loop_wafer</td>
<td>int</td>
<td>Current wafer loop flag</td>
<td>Any UAP except for lot end</td>
<td>1</td>
</tr>
</tbody>
</table>

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 to Get Value” column.

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

If the variable type is string, you can view its value by using the following code in a UAP routine:

```python
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.

```python
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 what a test ends and the test data resets to {} when a test begins.

- ACS_wdf_nonexist
  - Type: dictionary
  - Structure: {(x, y):Site_pxny, ...}
  
  Example:
  
  {(1, 3): 'Site_p1p3', (3, 3): 'Site_p3p3'}

- ACS_wdf_head
  - Type: dictionary
  - Structure: {

"version" : string version,
"file" : string filename,
"date" : string date,
"comment" : string comment,
"project" : string Single,
"diameterunits" : string Metric|English,
"diameter" : float diameter,
"units" : string Metric|English,
"squarewaferx" : float width, wafer width in square wafer,
"squarewafery" : float height, wafer height in square wafer,
"diesizex" : float diesizex, or Glass width in LCD,
"diesizey" : float diesizey, or Glass length in LCD,
"orientation" : (string Flat|Notch, string Top|Bottom|Left|Right, string Top|Bottom|Left|Right),

"waferoffset" : (float xoffset, float yoffset),

"axis" : int 1|2|3|4,

"origin" : (int xdistance_to_target, int ydistance_to_target),

"target" : (int targetx, int targety),

"autoalignlocation" : (float alignx, float aligny),

"optimize" : int 0-9,

"siteusage" : int usage,

"chipsperreticle" : (float, float),

"reticletarget" : (float, float), or Mark1 position in LCD

"reticleoffset" : (float, float), or Mark2 position in LCD

"partial": int 0-1, or whether to show coordinate on glass map in LCD

"margin": float,

"color": {"default": string, "margin": string, "target": string, "target selected": string,

"current": string, "touched": string, "pass": string, "fail": string},

"numx": int Site num in x direction, added for LCD

"numy": int Site num in y direction, added for LCD

"squareflag": int 0-1, default 0

"userdefvalue1" : string,
"userdefvalue2" : string,

:

:

"userdefvalue10" : string

}

Example: {'comment': '',

'diameter': 8.0,

'partial': 0,

'orientation':('Flat','Left','Bottom'),

'squareflag': 0,

'color': {'targetsel': (255, 255, 0),

'target': (255, 0, 0),

'default': (129, 243, 235),

'invalid': (193, 193, 193),

'current': (0, 0, 255),

'grid': (157, 149, 130),

'touched': (0, 255, 0),

'pass': (0, 255, 0),

'fail': (255, 0, 0),

'margin': (171, 168, 217)},

'optimizepriority': 'die',

'userdefvalue10': '',

'file': '',

'axis': 2,

'diameterunits': 'English',

'scale': 1,

'autoalignlocation': (0.0, 0.0),
'siteusage': 0,
'userdefvalue6': ",
'userdefvalue7': ",
'userdefvalue4': ",
'userdefvalue5': ",
'userdefvalue2': ",
'userdefvalue3': ",
'version': 1.1000000000000001,
'userdefvalue1': ",
'units': 'Metric',
'reticuleoffset': (0.0, 0.0),
'userdefvalue8': ",
'userdefvalue9': ",
'logicrow': 1,
'numx': 4,
'numy': 4,
'diesizey': 20.0,
'logiclist': [(0, 0)],
'reticletarget': (0.0, 0.0),
'date': ",
'diesizex': 20.0,
'optimize': 0,
'origin': (5, 5),
'movestepdiv': 10,
'target': (0, 0),
'solid': 1,
'waferoffset': (0.0, 0.0),
'project': 'Single',
'logiccol': 1,
'chipsperreticle': (0.0, 0.0),
'squarewaferx': 100.0,
'squarewafery': 100.0,
'margin': 2}

- ACS_temp_klf
  - Type: string

Usage: Defines the location of a temporary limits file. You can construct a Keithley limits file (.klf) object and get limits information. The .klf object has two common properties: head and limits. Head is dictionary type and limit is a list type.

.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))

**Global functions**

Functions that can be accessed in UAP PTMs are listed in the next Table:
Table 2: ACS Global Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_SET_GLOBAL_VALUE(name, value)</td>
<td>Update global variable with value. Only this function can change a global variable value</td>
</tr>
<tr>
<td>FUNC_GET_GLOBAL_VALUE(name)</td>
<td>Get a global variable’s value.</td>
</tr>
<tr>
<td>FUNC_SET_USER_GLOBAL</td>
<td>Define new variable in user global data pool. Only used between UAP</td>
</tr>
<tr>
<td>FUNC_SYS_GLOBAL(name, value)</td>
<td>Set system global variable value.</td>
</tr>
<tr>
<td>FUNC_EXIT_KTXE</td>
<td>Exit ktxe execution</td>
</tr>
<tr>
<td>FUNC_SET_KTXE_SUBSITES</td>
<td>Set subsites list used by the execution engine.</td>
</tr>
<tr>
<td>FUNC_SET_KTXE_LOOP_WAFER</td>
<td>Set wafer loop flag</td>
</tr>
<tr>
<td>FUNCTION_EXEC_TEST</td>
<td>Execute the specified test module.</td>
</tr>
<tr>
<td>FUNC_GET_KDF_HEADER</td>
<td>Get kdf header. Returns data as a dictionary.</td>
</tr>
<tr>
<td>FUNC_SAVE_KDF_HEADER</td>
<td>Save kdf header dictionary into a kdf file.</td>
</tr>
<tr>
<td>FUNC_SAVE_KDF_WAFERID</td>
<td>Save kdf wafer id into a kdf file.</td>
</tr>
<tr>
<td>FUNC_SAVE_KDF_EOW</td>
<td>Save kdf wafer delimiter (&lt;EOW&gt;) into a kdf file.</td>
</tr>
<tr>
<td>FUNC_SAVE_KDF_SITEID</td>
<td>Save kdf site id into a kdf file.</td>
</tr>
<tr>
<td>FUNC_SAVE_KDF_EOS</td>
<td>Save kdf site delimiter (&lt;EOS&gt;) into a kdf file.</td>
</tr>
<tr>
<td>FUNC_GET_KDF_DATA</td>
<td>Get kdf data for the specified wafer and site id.</td>
</tr>
<tr>
<td>FUNC_GET_NEW_KDF_DATA</td>
<td>Get the latest kdf data for the specified wafer and site id.</td>
</tr>
</tbody>
</table>
### Get Object Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FUNC_SAVE_KDF_DATA</td>
<td>Save kdf data into a kdf file.</td>
</tr>
<tr>
<td>FUNC_SITE_COORD_2_ID</td>
<td>Translate site coordinate to site id.</td>
</tr>
<tr>
<td>FUNC_SITE_ID_2_COORD</td>
<td>Translate site id to site coordinate.</td>
</tr>
<tr>
<td>FUNC_GET_WDF_OBJ</td>
<td>Get the wdf object used by the execution engine.</td>
</tr>
<tr>
<td>FUNC_GET_KDF_OBJ</td>
<td>Get kdf object from the test tree used by the execution engine.</td>
</tr>
<tr>
<td>FUNC_GET_SUBSITE_OBJ</td>
<td>Get the subsite object from the test tree used by the execution engine</td>
</tr>
<tr>
<td></td>
<td>according to the subsite name.</td>
</tr>
<tr>
<td>FUNC_GET_DUT_OBJ</td>
<td>Get the DUT object from the test tree used by the execution engine</td>
</tr>
<tr>
<td></td>
<td>according to the DUT’s name.</td>
</tr>
<tr>
<td>FUNC_GET_TEST_OBJ</td>
<td>Get the test object from the test tree according to the test name.</td>
</tr>
</tbody>
</table>

### Get Test Info Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET_EXEC_TEST_DIC</td>
<td>Get content of the test dictionary</td>
</tr>
<tr>
<td>GET_EXEC_TEST_LIST</td>
<td>Get test list. It is the key in test dictionary</td>
</tr>
<tr>
<td>FUNC_GET_DUT_NAME</td>
<td>Get the DUT’s name.</td>
</tr>
</tbody>
</table>

Details of the functions are described below:

**FUNC_SET_GLOBAL_VALUE(Name, Value)**

Function:

Update global variable with value.

Only this function can be used to change a global variable’s value.

Parameter:

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

Value: string.
Example:

```python
FUNC_SET_GLOBAL_VALUE('ACS_wafer_id', 'wafer_1')
```

**NOTE**

A variable must have quotation marks. For example, ‘ACS_wafer_id’ or “wafer_1.”

**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:**

```python
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:**

```python
g_site_id_list = []
```

# g_site_id_list must be declared in advance

```python
FUNC_SET_USER_GLOBAL("g_site_id_list")
```
**FUNC_SET_KTXE_LOOP_WAFER(Value)**

Function:

Set the ACS_ktxe_loop_wafer global variable which is the current wafer loop flag. If 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(subsitelist)**

Function:

Change the subsite list by the execution engine. Used in 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 kdf exists, False, If kdf does not exist.

Example: Use this function at lot begin UAP

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

}
'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:

\texttt{FUNC\_SAVE\_KDF\_WAFERID('C://a.kdf', '01')}

\textbf{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:

\texttt{FUNC\_SAVE\_KDF\_EOW('C://a.kdf')}

\textbf{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', data);`
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:

```python
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])
```

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:

```python
import LogManager as log

d.log.logMessage(str(FUNC_SITE_COORD_2_ID((1,-2))))
```

**FUNC_SITE_ID_2_COORD (siteid)**

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:

```python
import LogManager as log

d.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:

```python
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:**

```python
(('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.

**Example:**

```python
import LogManager as log

log.logMessage(str(FUNC_EXEC_TEST(('03', 'Pattern_1', 'Site_n2p2', 'subsite', 'Resistor_1k', 'sweepv'))))
```

**OUTPUT:**

```json
{'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:

```python
import LogManager as log

log.logMessage(str(FUNC_GET_TEST_OBJ(ACS_test_id)))
```

Output:

```
<testtree.TEST instance at 0x064CC6E8>
```

```python
log.logMessage(str(FUNC_GET_TEST_OBJ(ACS_test_id).limit))
```

Output:

```
[{'Target': 0,
'ValidHigh': 9.9999999999999997e+098,
'ConsFail': 0,
'ValidLow': -9.9999999999999997e+098,
'Critical': 0,
'Exit': 'None',
'SpecLow': -9.9999999999999997e+098,
'Report': 1,
'Sigma': 1.0,
'Unit': 'A',
'SpecHigh': 9.9999999999999997e+098}, ...]
```

**FUNC_GET_SUBSITE_OBJ(subsite_name)**

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.siteList
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:

```python
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:**

```python
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:

```json
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>

Useful tools for UAP routines

Importing python modules

Defined modules can be reused by another program by using the import command. There are three 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.

Useful 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.

\[
\text{atoid(str, base = 10)}: \text{Convert a string into integer using the optional base(default is decimal). The difference between this function and Python standards string.atoid(x[,base]) is, this function allows you to input a string with characters other than decimal digits. atoi attempts to parse as many characters as it can to build an integer.}
\]

\[
\text{atof(val): Convert a string into float. The difference between this function and Python standards string.atof(x) is, this function does not force you to input a pure float. atof() attempts to parse as many characters as it can to build a float.}
\]

\[
\text{isnan(num): whether num is -1.#IND}
\]

\[
\text{getSetting(filename, section, item, openc = '}', closec = '{', asignc = '='}, multi_asignc = False, matchFlag = 0): operation for ini file, Extract a setting from a file. The file has a format similar to}
\]

\[
\text{[Section 1]}
\text{Item1=Setting 1}
\text{Item2=Setting 2}
\text{Item3=Setting 3}
\]

\[
\text{[Section 2]}
\text{Item1=Setting 1}
\text{Item2=Setting 2}
\]

\[
\text{putSetting(filename, section, item, value, openc = '}', closec = '}', asignc = '='}: an operation for ini file, corresponding to getSetting, put the section's item value.
\]

\[
\text{delSetting(filename, section, item, openc = '}', closec = '}', asignc = '='}: an operation for ini file, delete the section's item.
\]

\[
\text{avgsdev(data): Get average and standard deviation of data.}
\]
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.

Useful 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 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. An example screenshot is showed below.

Figure 104: Sample `help()` description

- `deepcopy(obj)`: Deep copy operation on arbitrary Python objects. Mostly used with recursive objects(object contains another object)
- **numpy**
  
  \texttt{sin()}: get sin value
  
  \texttt{pi}: const pi
  
  \texttt{array}: Return an array from object with the specified data-type.
  
  \texttt{sum}: Sum the array over the given axis.

- **os**
  
  \texttt{os.path module provides many useful common pathname manipulations. Please see help(os.path) for details in a PythonWin window.}
  
  \texttt{os.path.join(a, \_\_p\_\_): Join two or more pathname components, inserting "\_\_" as needed.}
  
  \texttt{os.mkdir(path): Create a directory. If an intermediate directory doesn't exist, an error is raised.}
  
  \texttt{os.makedirs(path): Super-mkdir. Recursive create a directory. If a directory doesn't exist, create it.}

- **scipy**
  
  \texttt{min, max, median, average, stddev}

- **shutil**
  
  \texttt{shutil.copy2(srcname, destname): Copy data and all state info}

- **string**
  
  \texttt{string.atof(s): Return the floating point number represented by the string s.}

- **time**
  
  \texttt{time.sleep(t): Delay execution for a given number of seconds.}
  
  \texttt{time.strftime (format[, tuple]): Return format string. For example:}
  
  \texttt{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 \texttt{help(types)} in PythonWin after you import types.

- **xls**: Excel module with xml format. For additional detail, use the \texttt{help(xls)} in PythonWin after you import xls

### Useful modules in wxpython

- \texttt{wx.MessageDialog(): Popup message window.}
- \texttt{wx.TextEntryDialog: Popup text entry dialog, get the input value.}
- \texttt{wx.grid:}

### Useful .dll module

- \texttt{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; to monitor the test process or to write a user data file. The following describes how to use 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 pylibrary, that show how to call a wxpython dialog from a UAP routine.

Write user data file

There are two example files provided that shows how to write data to a file from a UAP routine. Refer to the code in ChooseFileToSave.py and/or site_ACS_to_file.py for additional details.