tek.com/keithley

S540 Power Semiconductor Test System

Reference Manual

S540-901-01 Rev. B / January 2019

S540-901-01B

KEITHLEY
A Tektronix Company

S540

Power Semiconductor Test System

Reference Manual

Cleveland, Ohio, U.S.A.

These are the original instructions in English.

Document number: S540-901-01 Rev. B / January 2019

Safety precautions

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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

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

Safety precaution revision as of June 2017.

Table of contents

Section 1

Introduction

In this section:

Contact information

If you have any questions after you review the information in this documentation, please contact your local Keithley Instruments office, sales partner, or distributor. You can also call the corporate headquarters of Keithley Instruments (toll-free inside the U.S. and Canada only) at 1-800-935-5595, or from outside the U.S. at +1-440-248-0400. For worldwide contact numbers, visit the Keithley Instruments website (tek.com/keithley).

Systems documentation

Documentation for your system is available at tek.com/keithley. Following is a list of documentation for your system, including the document part numbers.

- S540 Power Semiconductor Test System Administrative Guide (S540-924-01)
- S540 Power Semiconductor Test System Reference Manual (S540-901-01)
- Keithley Test Environment (KTE) Programmer's Manual (S500-904-01)

System description

The Keithley Instruments S540 Power Semiconductor Test System is a configurable, instrument-based system for power semiconductor parametric characterization and testing. There are two different S540 systems available:

- S540 3000 V high-voltage parametric test system with a 3-kV switching matrix (12 pins)
- S540 3000 V high-voltage, low-current parametric test system with a 3-kV switching matrix (12 pins) and a low-current switching matrix (12, 24, or 36 pins)

The S540 systems have flexible hardware configurations that allow you to customize them to your specific needs. See the following table for a description of the main system configuration options, and see the following figure for an example of a typical system configuration.

S540 3000 V high-voltage system configuration options

DC source-measure units (SMU)	■ Maximum number of SMUs depends on other items in the system rack ■ Model 2636B System SourceMeter® instrument (quantity: 0 to 4) ■ Model 2657A High-Power System SourceMeter® instrument (quantity: 2)
Switching matrices	One Model HVM1212A 12 × 12 high-voltage switch matrix
Optional capacitance/voltage (C-V)	One channel of C-V (using one Model 4200A-SCS with one Model 4210-CVU card)
Optional instruments	■ Model 4220-PGU dual-channel pulse cards (quantity: 0 to 3) ■ RSA306B USB Spectrum Analyzer (quantity: 0 to 1) for frequency and ring oscillator measurements
Included with each system	■ Computer inside cabinet ■ External 24-inch flat-panel monitor and keyboard tray mounted on exterior of cabinet ■ Keithley Test Environment (KTE) or Automated Characterization Suite (ACS) system software ■ LO patch panel ■ Safety interlock system ■ Adjustable cable support arm
Other options	Advanced seismic securement kit for additional resistance to seismic forces

S540 3000 V high-voltage, low-current system configuration options

DC source-measure units (SMU)	■ Maximum number of units depends on other items in the system rack ■ 2636B System SourceMeter® instrument (quantity: 1 to 4) ■ 2657A High-Power System SourceMeter® instrument (quantity: 2)
6-slot switching matrix	One Model HVM1212A 12 × 12 high-voltage switch matrix and one Model 707B system switch mainframe with: ■ Model 7531 switch cards (quantity: 2 to 6, depending on instruments and number of low-voltage pins in the system); one card is used as an interconnect card
Optional capacitance/voltage (C-V)	Two channels of C-V (using one Model 4200A-SCS with two Model 4210-CVU cards)

S540 3000 V high-voltage, low-current system configuration options

Optional instruments	■ Model DMM7510 7-1/2 Digit Graphical Sampling Multimeter (quantity: 0 to 1) ■ Model 4220-PGU dual-channel pulse cards (quantity: 0 to 3) ■ Frequency measurement option
Included with each system	■ Computer inside cabinet ■ External 24-inch flat-panel monitor and keyboard tray mounted on exterior of cabinet ■ Keithley Test Environment (KTE) or Automated Characterization Suite (ACS) system software ■ LO patch panel ■ Interlock system ■ Adjustable cable support arm
Other options	Advanced seismic securement kit for additional resistance to seismic forces

Optional accessories

Optional items and accessories that may accompany the S540 system:

- Cables to connect to the test fixture or the probe card adapter
- 9140A-PCA probe card adapter (12 3-kV pins to 36 200-V pins) or Celadon Systems VersaCore™ VC20 probe card adapter (12 3-kV pins to 32 200-V pins)
- RSA306B USB Spectrum Analyzer
- Advanced seismic securement kit

When you receive your system

Series 500 systems are shipped in a wooden crate (see the following figure).

Figure 1: S540 system cabinet in shipping crate

Accessories are shipped in a separate box.

Inspect for damage

NOTE

The field service engineer (FSE) is responsible for unpacking the crate. If the crate is damaged when you receive it, the FSE will do a thorough inspection of each of the system components.

Inspect the shock sensor on the outside of the shipping crate (see the following figure). If the shock sensor indicates a shock condition, do a thorough inspection of all components in the system cabinet.

Figure 2: S540 crate shock sensor

Also, check the "TIP N TELL" indicator to ensure that the crate has not been tipped over (see the following figure).

Figure 3: S540 crate tipping indicator

Carefully remove all system components from the crate. While unpacking, make sure there is no component damage. Report any damage to the shipping agent immediately.

Move the shipping crate to the installation location

We recommend that you move the crate and the accessories box to the area where the system is going to be used, but do not unpack them. The Keithley field service engineer (FSE) is responsible for unpacking the S540 system cabinet and the accessories. For more information about unpacking, see the *S540 Administrative Guide* (part number S540-924-01).

Section 2

Overview of system instruments

In this section:

Introduction

This section contains an overview of the instruments used in S540 test systems and examples of typical connection schemes.

For more specific information about instruments used in the S540 Power Semiconductor Test System, refer to the documentation for each specific model:

- 4200A-SCS Parameter Analyzer
- Series 2600B System SourceMeter® Instrument
- 4210-CVU Capacitance-Voltage Unit Card
- 4220-PGU High-Voltage Pulse-Generator Unit Card
- RSA306B Signal Analyzer
- 707B Semiconductor Switch Matrix
- 2657A High-Power System SourceMeter Instrument
- DMM7510 7-1/2 Digit Graphical Sampling Multimeter

Refer to the documentation that is bundled with the Keithley Test Environment (KTE) software installation. You can also visit the Keithley Instruments website at tek.com/keithley to search for updated information by model number.

NOTE

Example wiring diagrams for the S540 test system are shown later in this section.

Switching

The following components provide the switching capabilities of the S540.

Model 707B Switch Matrix Mainframe

The 707B Semiconductor Switch Matrix Mainframe, which is included in S540 3000 V, low-current systems, is a programmable switch for connecting signal paths in a matrix structure. The six-slot mainframe accepts any combination of compatible plug-in matrix cards. Model 7531 matrix cards are used in the 707B in S540 systems.

Figure 4: 707B Semiconductor Switch Matrix Mainframe

Model 7531 Low-Current, High-Speed Matrix Card

The 7531 matrix card is an 8 × 12 low-current, high-speed Kelvin matrix with high-voltage protection up to 3 kV. The S540 3000 V high-voltage, low-current systems include one 707B System Switch Mainframe populated with 7531 matrix cards.

The 7531 matrix card allows high-voltage source-measure units (SMUs) to provide output to low-voltage circuits without damage. The high-voltage pass-through signals are clamped at a safe level by the protection modules in the circuit.

HVM1212A High-Voltage Switch Matrix

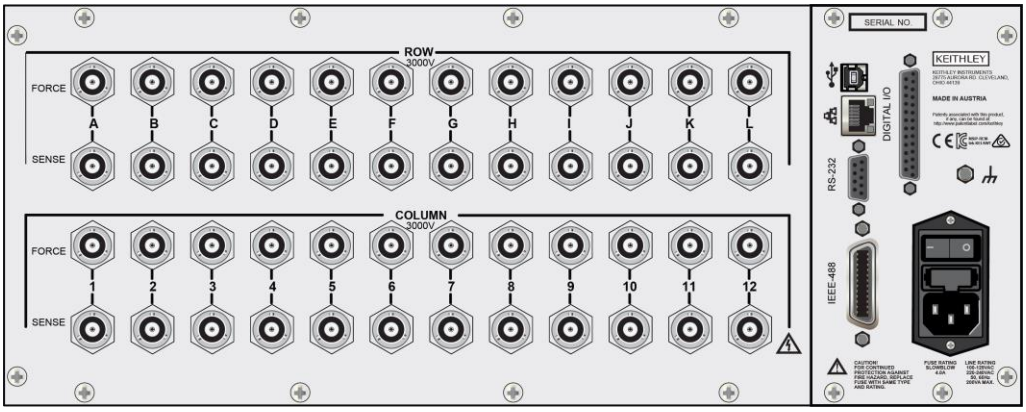
The HVM1212A is a 12 × 12 high-voltage switching matrix that can handle signals up to 3 kV. One HVM1212A High-Voltage Switch Matrix is included in both the S540 3000 V high-voltage test system and the 3000 V high-voltage, low-current test system.

Figure 5: HVM1212A High-Voltage Matrix

In 3000 V high-voltage, low-current systems, the HVM1212A connects pass-through signals to the 707B low-current matrix through protection modules to one of the 7531 matrix cards. This allows you to do high-voltage, low-voltage, low-current, and capacitance tests in a single probe touch-down.

The following figure shows the connections on the rear panel of the HVM1212A.

Figure 6: HVM1212A high-voltage switch matrix rear panel



Sourcing and measuring

The following instruments provide S540 source-measure capabilities.

Model 2636B System SourceMeter Instrument

The 2636B Dual-Channel System SourceMeter® Instrument is a 6-1/2 digit, two-channel source-measure unit (SMU) that simultaneously sources and measures voltage and current. You can have up to four 2636B SMUs (for a total of eight SMU channels) in the S540 system.

Figure 7: 2636B System SourceMeter Instrument

Features include:

- Maximum voltage source-measure range: 200 V
- Measurement resolution: 0.1 fA current, 100 nV voltage
- Power: 20 W

Model 2657A High-Power System SourceMeter Instrument

The Model 2657A is a high-voltage, high-power, low-current source measure unit (SMU) instrument. You can have up to two 2657A SMUs in the S540.

Figure 8: 2657A High-Power System SourceMeter Instrument

Features include:

- Source ± DC voltage from 5 mV to 3030 V
- Source ± DC current from 30 fA to 120.12 mA
- Measure ± DC voltage from 1 mV to 3030 V
- Measure ± DC current from 20 fA to 120.12 mA

Optional instrumentation

Keithley Test Environment (KTE) version 5.7.0 supports several optional instruments:

- DMM7510 7½ Digit Graphical Sampling Multimeter
- 4200A-SCS Parameter Analyzer
- 4210-CVU Capacitance-Voltage Unit Card
- 4220-PGU High-Voltage Pulse Generator Unit Card
- RSA306B USB Spectrum Analyzer

NOTE

The cards and columns of each instrument connection shown in the following diagrams may differ from your actual system. The flexibility of the S540 configuration allows for various numbers and combinations of instruments. Attempting to show examples of every possible scenario would be prohibitive.

For high-voltage systems with two 707B switch matrices, the high-voltage connections for the second matrix (not shown) are identical to the ones in the diagrams in this manual.

Section 3

Installation

In this section:

Site preparation

The following topics will help you get your site ready for installation.

Site preparation checklist

NOTE

The following site preparation checklist will help you prepare your site for the S540 system in your facility. If you find that an item listed is not valid for your site, you can indicate it with "N/A."

S540 system preparation checklist

Site	**Item**
	Is it necessary to have lifting equipment?
	Is the flooring adequate and able to support the weight of the system while moving from receiving to the final destination (see Floor plan (on page 3-3) for specifications)?
	Are all of the corridors and hallways large enough to allow clearance for the system?
	Are stairways adequate for moving the system through?
	Are elevators needed to move the system? Can they support the size and weight of the system?
	Are the doorways wide enough for the system?
	If you are using a Keithley probe card adapter, you must supply a vacuum connection (50.80 cm Hg / 20 in.).
Floor plan	**Item**
	Did you complete the system layout (see Floor plan (on page 3-3) for specifications)?
	Does your layout show all of the locations for all of the equipment?
	Does your system layout show the locations of all doors and aisles?
	Does your layout allow for the proper clearance of the system for the front, rear, and the keyboard/monitor arm?
	Is there enough space for personnel safety, comfort, and freedom of movement?

S540 system preparation checklist (continued)	
Floor plan	**Did you take future expansions into consideration?**
Electrical power	**Item**
	Is there sufficient space for any supplies or manuals?
	Is adequate and proper electrical power available (see Line power requirements (on page 3-2) for specifications)?
	Is anything connected to the same power source that generates noise?
	Is anything that requires substantial amounts of current connected to the same power source?
	Did you prepare power outlets for service, testing, or maintenance?

Line power requirements

Nominal input line voltage: 100 VAC, 115 VAC, 220 VAC, 240 VAC (50 Hz, 60 Hz)

Short-circuit current rating: 5 kA

Power consumption: Rated at 2.4 kVA for the 2 kW power distribution unit (PDU)

Heat generation: Quiescent heat of 1720 BTU (1815 kJ) to maximum heat of 8191 BTU (8642 kJ)

⚠ WARNING

Severe personal injury or death due to electric shock or electrocution or equipment damage may occur if you do not have the correct circuit amperage.

S540 systems that are configured to operate between 100 VAC and 120 VAC must use a 20 A circuit; systems that are configured to operate between 200 VAC and 240 VAC must use a 15 A circuit.

Operating environment conditions

To ensure operation within specifications, the S540 must be operated inside of the following environmental conditions.

Temperature: 23 °C ±5 °C

Operating humidity: 30% to 60% relative humidity, noncondensing, after a two-hour warm up time

Vibration: High ambient vibration levels may require isolation pads or the repositioning of equipment

Air quality: The S540 system is compatible for use in a Class 10 clean room

Audible system noise: Decibel level is 65 dBA in optimal environmental conditions

Airflow: The S540 system is configured for top to bottom airflow

Altitude: Less than 2000 m (6,561 feet) above sea level

Noise interference: To prevent electrical noise from interfering with measurements, the ambient AC magnetic field must not exceed 2×10^{-3} G (2×10^{-7} T):

- Avoid locating the S540 next to plasma etchers, large motors, magnets, RF transmitters, equipment with flash lamps, and other potential sources of interference
- Position equipment to avoid routing signal and power cables near sources of electrical noise

Floor plan

NOTE

The following floor plan information is for the system cabinet only. Be sure to place the cabinet a minimum distance of 15 cm (6 in.) up to a maximum distance of 122 cm (48 in.) from the prober. Refer to the documentation for the prober or other test-fixture equipment to determine its floor space requirements.

The system cabinet requires a floor space of approximately 1.2 m × 2.1 m (4 ft × 7 ft), plus additional room for service personnel access from the front and back of the cabinet. The following figures show a top view of the floor plan and the typical S540 system cabinet weight distribution and center of gravity. System cabinet size and weight (on page 3-5) lists the dimensions and weight of the system cabinet.

See the figure on the following page for floor plan specifications.

Figure 9: S540 floor plan, top view

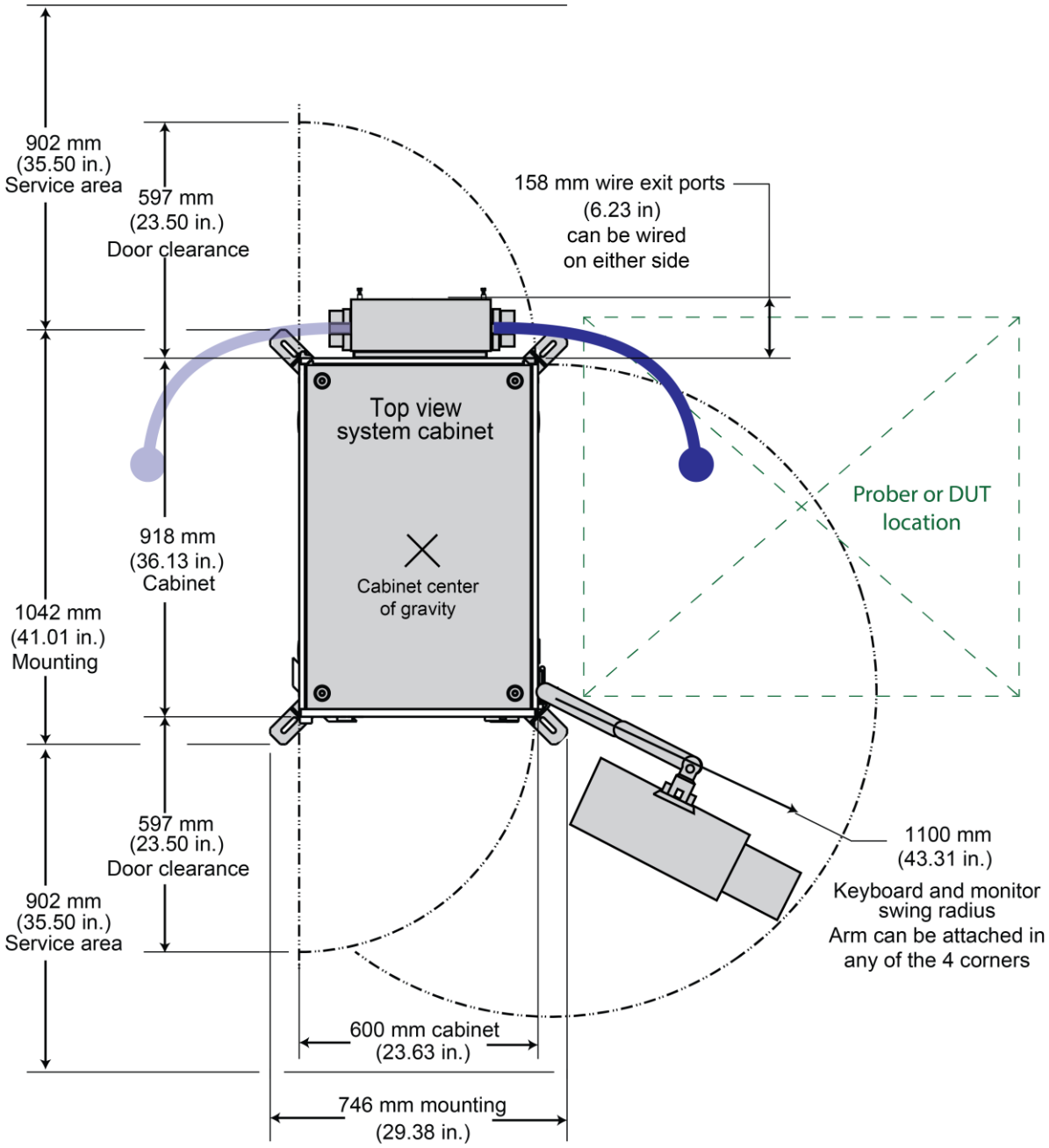
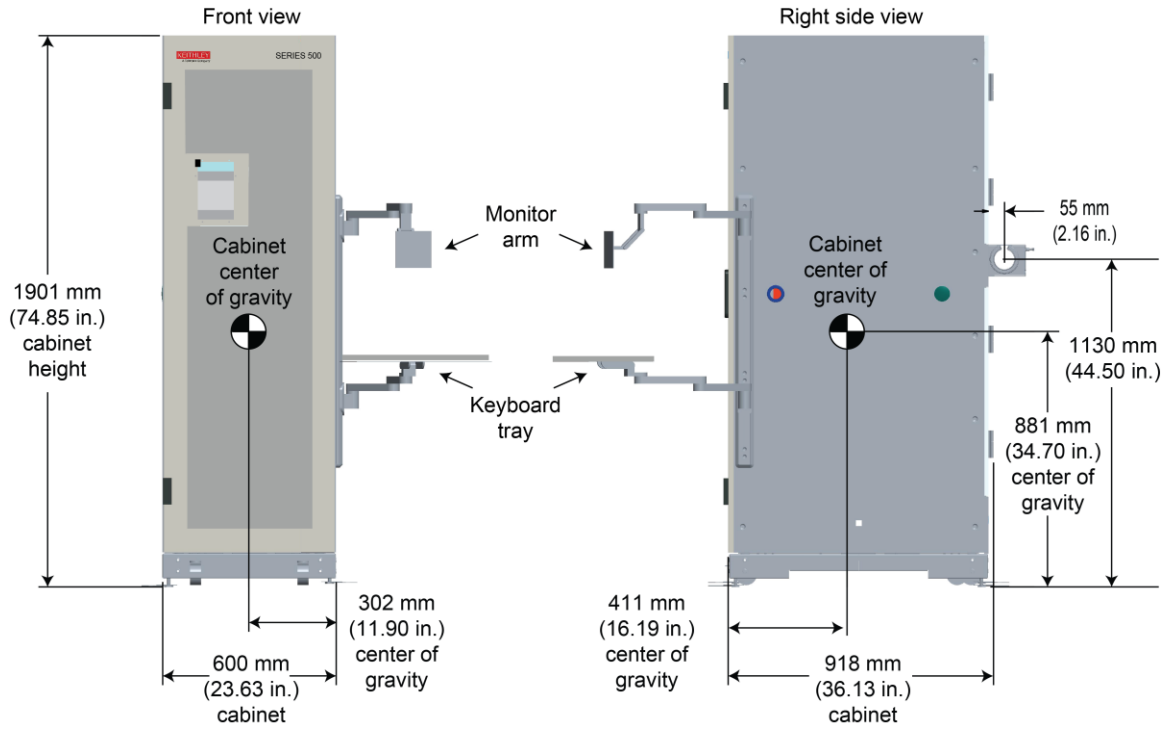


Figure 10: S540 weight distribution and center of gravity



System cabinet size and weight

The size and weight specifications for the system cabinet are listed in the following table. See Floor plan (on page 3-3) for details about designing a floor plan for the system cabinet.

System cabinet size and weight						
Size (width × depth × height)	**Weight**					
	Minimum configuration			**Maximum configuration**		
600 mm × 918 mm × 1905 mm (23.63 in. × 36.13 in. × 75.00 in.)	195.05 kg (430 lb)			396.89 kg (875 lb)		
	Front feet	Left	51.83 kg (114.25 lb)	Front feet	Left	105.46 kg (232.50 lb)
		Right	51.83 kg (114.25 lb)		Right	105.46 kg (232.50 lb)
	Rear feet	Left	45.70 kg (100.75 lb)	Rear feet	Left	92.98 kg (205 lb)
		Right	45.70 kg (100.75 lb)		Right	92.98 kg (205 lb)

Required system securement

You must bolt the S540 system cabinet to the floor for safety purposes and to ensure the cabinet will not tip over.

If you are in an area that has seismic activity, an optional advanced seismic securement option is available.

For more detailed information about securing the system to the floor, see the *S540 Administrative Guide* (part number S540-924-01).

Keithley field service engineer installation tasks

The Keithley field service engineer (FSE) will perform the following tasks:

- Unpack the system components and accessories.
- Attach the keyboard arm and monitor arm to the system.
- Install the keyboard and the mouse on the keyboard arm, and the monitor on the monitor arm.
- Install the cable support arm to the system.
- Install the probe card assembly (PCA) (if ordered) on the back of the system cabinet, and the 60190-PCA (probe card assembly) to the correct prober plate (customer-supplied from the prober company). The prober plate is attached to the prober.
- Plug in the system to your power facilities (supplied by your facilities department at the final location for the S540 system cabinet) and power up the entire cabinet.
- Verify communications of all instruments and with the properly configured prober.
- Perform diagnostics and system verification tests of the entire S540 system, to include the 60190-PCA (if ordered).
- Record all the information on the System Installation Form.

System connections

All connections for your system are made by the Keithley field service engineer (FSE) when your system is installed.

For detailed drawings of matrix connections and communications diagrams, refer to the *S540 Administrative Guide*, part number S540-924-01.

Section 4

Getting started

In this section:

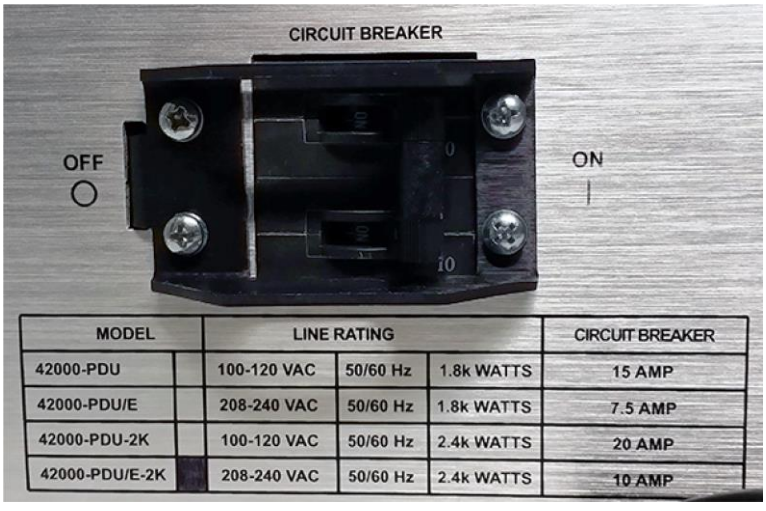
Initial equipment startup

To begin equipment startup:

1. Check that all power cords for the system cabinet are connected to AC power.

   Make sure that the circuit breaker on the power distribution unit (PDU) is in the ON position.

Figure 11: S540 power distribution unit (PDU) circuit breaker



2. Press the power/standby button on the computer and monitor.

3. Set the power button on the front door of the system cabinet to the ON position.

Figure 12: S540 power ON switch

System startup

To start up the system:

1. Make sure that the power switch on the power distribution unit (PDU) is set to ON.
2. Set the power button on the front door of the system cabinet to the ON position.
3. If the computer has not started to boot, open the front cabinet door and press the power/standby switch on the host computer.
4. Wait for all of the instruments to power up.
5. Log onto your computer.

System software

ATTENTION

To avoid instrument errors, make sure that all of the instruments in the system are completely powered up and have finished self-testing before starting the system software.

The S540 includes one of the following system software options:

- Keithley Test Environment (KTE)
- Automated Characterization Suite (ACS)

For more information about the KTE software, see Software (on page 6-1).

For more information about the ACS software, see the *Automated Characterization Suite (ACS) Reference Manual* (part number ACS-901-01).

NOTE

Telnet is not enabled by default on the S540 system; you must to enable it to use it. For details about enabling Telnet, see "Using Telnet" in the *S540 Administrative Guide* (part number S540-924-01).

Start the KTE software

To start the KTE software, first start the instrument controller (IC) process. To start the IC process, log on to the computer and enter the following command:

```
run_ic.pl
```

NOTE

The IC process must be started manually after you log on.

The first time the IC process is started after a new KTE software installation, all instruments in the TSP-Link® network are initialized, which may take a few minutes. A status message is displayed on the TSP master instrument (Model 2450) during this process.

After the initial software installation, `run_ic.pl` initializes only the TSP-Link master instrument, and the other TSP instruments in the system execute the appropriate TSP script without delay.

Shut down using KTE

To shut down using the Keithley Test Environment (KTE) software:

1. Close all KTE programs.
2. In the LINUX® terminal, type the following command:
   `sudo $KIBIN/shutdown_s530.pl`
3. Enter the requested password.
4. Wait for the system computer to stop.
5. Turn off power to the cabinet.

Emergency OFF button

An EMERGENCY OFF (EMO) button is on the system cabinet front door (see the following figure). If you push the EMERGENCY OFF button, it removes power to all of the system instruments except the host computer.

Figure 13: EMERGENCY OFF button

The EMO TRIPPED indicator light (on the cabinet door) turns on when the system has undergone an emergency shutdown.

Emergency shutdown procedure

Press the red **EMERGENCY OFF** button on the front of the system cabinet. The instruments power down and the red EMO TRIPPED indicator illuminates.

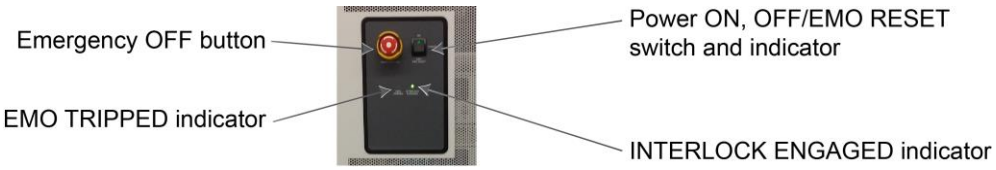
The red indicator also illuminates when the system recovers from a sudden power loss.

Recovering from an emergency shutdown

To recover after an emergency shutdown:

1. Verify that the hazardous condition or emergency situation is no longer present.
2. Rotate the **EMERGENCY OFF** button to release it.
3. Toggle the power switch from ON to OFF, and then back to ON again. All of the system instruments should power up.
4. Open the front cabinet door and press the power/standby switch on the host computer.
5. Call `run_ic.pl` to reinitialize the instrumentation.

Figure 14: S540 front-panel controls and indicators



Safety interlocks

WARNING

Failure to make sure that the safety interlock and safety shields and guards are properly installed and arranged as indicated will put personnel in severe danger. Severe personal injury or death due to electric shock or electrocution may result.

For the safety interlock to function properly, the device under test (DUT) interlock sensor must be installed near the DUT connections and the interlock magnet must be installed on the safety shield. It must be set up so that when the magnet is near the switch (interlock closed) the operator cannot touch voltage-carrying conductors. If not properly installed, it will render the interlock inoperative and place personnel at severe risk.

For operator safety, the S540 has interlocks on both the front and back cabinet doors and at the DUT. Also, the optional probe card adapter (PCA) has interlocks that provide protection for connections to a prober.

If you open a cabinet door or open the DUT interlock while instruments are sourcing, the interlock activates and disconnects the hazardous voltage from the source-measure instruments, stopping any tests in progress.

An indicator on the front door of the S540 cabinet illuminates, and the KTE software immediately notifies you of the interlock activation.

Figure 15: Interlock indicator

Once the interlock has been activated, you must clear the cause of the interlock activation.

To clear the interlock activation:

1. Follow the instructions on the computer.
2. Make sure the front and rear doors are closed.
3. Make sure the DUT interlock is properly set for safe operation.
4. Close the DUT safety shield.
5. The software must recover before you can continue normal operation. You may need to rerun your tests.

Prober safety

WARNING

Hazardous voltages may be present on the probe card adapter, even after you disengage the interlock. Cables can retain charges after the interlock is disengaged, exposing you to live voltages that, if contacted, may cause personal injury or death.

Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

System communications

The following topics describe the communications connections for your system.

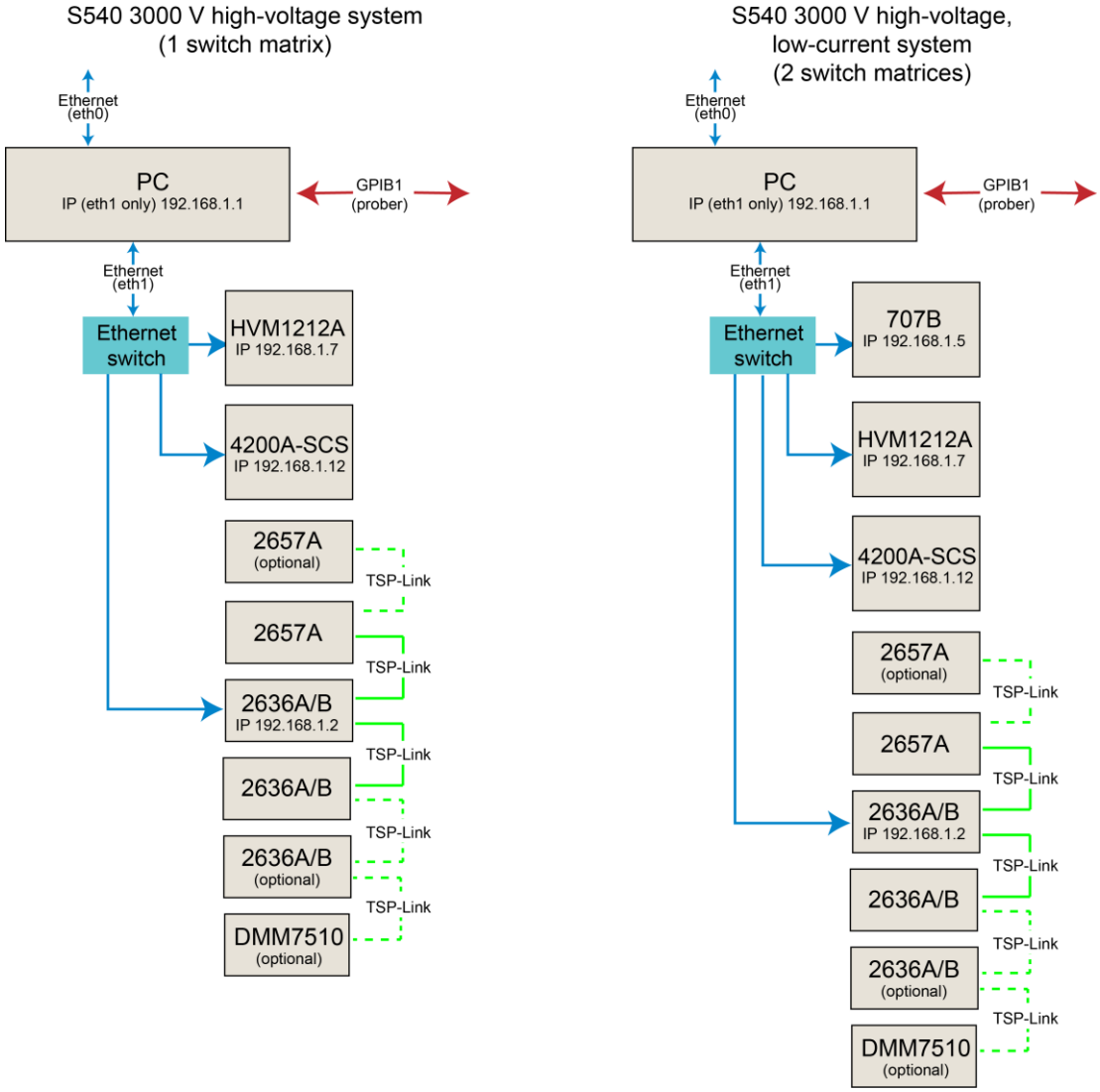
S540 KTE communications

The S540 KTE system uses both ethernet and GPIB to communicate with and control the instruments. The following diagrams show how the instruments are connected to each other and what type of communications are used.

NOTE

The following figure shows multiple 2636B source-measure units. However, the system only requires one 2636B.

Figure 16: S540 KTE communication diagrams



Editing the icconfig_<QMO>.ini file

The `$KIHOME/IC/icconfig_<QMO>.ini` file is a system configuration file that defines the types of instruments installed in the S540 system, matrix and terminal configuration, and default system settings. Your Keithley field service engineer (FSE) configures this file when your system is installed.

The FSE can edit this file to reflect changes in system hardware or to set different default system settings.

When editing the `icconfig_<QMO>.ini` file (where `<QMO>` is the system QMO number), follow the guidelines below:

- Make a backup copy of the file before making edits.
- Use only upper case characters.
- Do not use space or tab characters.
- Lines should be terminated with newline characters only (`\n`).
- Avoid duplicate entries and assignments.

Using NPLCs to adjust speed and accuracy

You can adjust the amount of time that the input signal is measured. Adjustments to the amount of time affect the usable measurement resolution, the amount of reading noise, and the reading rate of the instrument.

The amount of time is specified in parameters that are based on the number of power line cycles (NPLCs). Each power line cycle for 60 Hz is 16.67 ms (1/60); for 50 Hz, it is 20 ms (1/50).

The shortest amount of time results in the fastest reading rate, but increases the reading noise and decreases the number of usable digits.

The longest amount of time provides the lowest reading noise and more usable digits, but has the slowest reading rate.

Settings between the fastest and slowest number of PLCs are a compromise between speed and noise.

Setting the number of power line cycles

The default number of power line cycles (NPLCs) for measurements using the S540 can be set in several ways:

- Specify in the `icconfig_<QMO>.ini` file; your field service engineer (FSE) can change this from the factory default setting (see Setting a custom system speed (on page 4-9))

- Specify using the `setmode KI_INTGPLC` modifier (see "setmode" and "setmode modifier tables" in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01) for details); this setting does not persist through a power cycle

NOTE

The system default NPLC setting is `0.01`. However, this setting differs for some commands, source-measure unit (SMU) instruments, and ranges. See NPLC default differences (on page 4-10) for details.

Setting a custom system speed

NOTE

The `LEGACY` speed mode has been deprecated in KTE version 5.8.0. If you select this mode, the default system speed settings are used.

If you need different system speed settings than those set at the factory, you can ask your field service engineer (FSE) to specify custom speed settings in the `icconfig_<QMO>.ini` file. Systems upgraded from an older Keithley Test Environment (KTE) version may not have a `[SYSTEM SPEED MODE]` section in their `icconfig_<QMO>.ini` file. This can be added, but is not necessary unless you want to change the settings.

NOTE

The system default NPLC setting is `0.01`. However, this setting differs for some commands, source-measure unit (SMU) instruments, and ranges. See NPLC default differences (on page 4-10) for details.

To set a custom default system speed mode, the FSE can add the following block to the top of the icconfig_<QMO>.ini file (example):

```
[SYSTEM SPEED MODE]
MODE=CUSTOM
SOURCE_SETTLING=3
NPLC=0.01
ANALOG_FILTER=1
DELAY_FACTOR=0
```

NOTE

If you have problems with correlation of data using the system speed settings from the factory, your FSE can add the following line to your `icconfig_<QMO>.ini` file to force the system to use the older error mode: `TSP_ERROR_MODE OFF` (in KTE version 5.6.5 or later).

NPLC default differences

There are some instances where the number of power line cycle (NPLC) default values may be different than what is specified in the `icconfig_<QMO>.ini` file or `KI_INTGPLC` setmode modifier:

- The `meas`*`X`* command always measures with an NPLC of `0.01`.
- The default NPLC setting for the `intg`*`X`* and `sintg`*`X`* commands is `0.1` NPLC, except when using the 10 nA and lower ranges on a 2636B (default NPLC of `1`).
- The NPLC setting for measurements when running diagnostics is always `1`.

Default maximum voltage

The default maximum voltage for 2657A SourceMeter® Instruments (SMUs) in S540 systems is set to 3000 V in the `icconfig_<QMO>.ini`. You can change the default maximum voltage to any value from `300` to `3000` by changing the `MAX_HV` value in the `icconfig_<QMO>.ini` file. See Example icconfig_<QMO>.ini file with MAX_HV setting (on page 4-10) for an example.

This value can be temporarily overridden using the `setmode` LPT command modifier `KI_MAX_VOLTAGE` (see setmode and setmode modifier tables for details).

⚠ WARNING

Running system diagnostics or system verification overrides the MAX_HV setting in the `icconfig_<QMO>.ini` file; the system still sources up to 3000 V.

The Keithley Test Environment (KTE) software keeps track of forced voltage and limit values of the source-measure units (SMUs) to ensure that a voltage differential greater than 3000 V never gets forced. Attempting to force a differential voltage beyond 3000 V will result in an LPT error.

Call the `devint` command to restore the default specified in the `icconfig_<QMO>.ini` file.

Example icconfig_<QMO>.ini file with MAX_HV setting

```
[INSTRUMENTATION]
VERSION_ID=1
LIST=GNDU,HVGNDU,SMU1,SMU2,SMU3,SMU4,HVSMU1,HVSMU2,CMTR1,CMTR2,MTRX1,MTRX2
FULL_LIST=MTRX1,MTRX2,SMU1,SMU2,SMU3,SMU4,HVSMU1,HVSMU2,GNDU,CMTR1,CMTR2,VMTR1,KIB,
    KIP,GPT1,GPT2,PGU1A,PGU1B,SCP1A
DIAG=MTRX1,MTRX2,SMU1,SMU2,SMU3,SMU4,HVSMU1,HVSMU2
HARDWARE_ONLINE=TRUE
KELVIN_ENABLE=TRUE
PCA_INSTALLED=TRUE
PLC=60HZ
SWITCH=HV_HYBRID
MAX_HV=3000
```

Section 5

Using function libraries

In this section:

Introduction

The Keithley line of semiconductor test systems uses function libraries to control the instruments in the system. These libraries are called test control libraries. A test control library is the lowest level software interface to a parametric tester.

Keithley provides the following function libraries for use with S540 systems:

- **Linear Parametric Test Library (LPTLib):** Control commands for the most common S540 applications.
- **Parametric Test Subroutine Library (PARLib):** Test subroutines for parameter extraction and data analysis.
- **High-Voltage Library (HVLib):** Commands for two- and three-terminal measurement applications to measure capacitance-voltage (C-V), calculate compensation constants, do open, load, and short compensation, and do breakdown voltage tests.
- **Prober and Prober Driver User Library:** Commands for prober control.
- **Keithley Data Files (KDF) Library:** A set of routines to organize and save parametric test data into simple ASCII data files.
- **KTXE_RP Zone-Based Testing User Library:** Commands that randomly select sites to test at run time.
- **KTXE_AT Result-Based Testing User Library:** Commands to change the sites or tests to be used based on the results of previous site tests.
- **Keithley User Interface Library:** Commands to customize the user interface for operator data entry and program status monitoring.
- **Data pool commands:** Commands that control data stored in the data pool.

For detailed descriptions of the commands in these libraries, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

The KTE Linear Parametric Test Library (LPTLib) Library

The Linear Parametric Test Library (LPTLib) is one of the primary libraries that you can use with your S540 system. Following is an overview of the LPTLib commands that you can use with your system.

The following tables list the LPTLib commands with a brief description of each command. For detailed information about using the LPTLib commands, see the "LPTLib command reference" in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Combination commands

Command	Description
asweepX	Sweep with a user-defined force array (i, v).
bmeasX	Block measurement; take a series of readings as quickly as possible (i, v).
bsweepX	Sweep and shutdown source if device meets trigger condition (i, v).
clrscn	Clear any sweep measurements.
clrtrg	Clear any set triggers.
mpulse	Force a pulse and measure voltage and current.
rtfary	Return forced array after sweep.
savgX	Average each point of associated sweep (i, v).
searchX	Search for a specific current or voltage.
sintgX	Integrate each point of associated sweep (i, v, c, g).
smeasX	Measure each point of associated sweep (i, t, v).
sweepX	Sweep a specified range of current or voltage.
trigXg, trigXl	Trigger if a measurement is greater than a specific value (i, t, v).
	Trigger if a measurement is less than a specific value (i, t, v).

Dual-site commands

NOTE

These commands are only compatible with S535 test systems.

Command	Description
site_disable	Disable dual-site mode for the specified *siteid*.
site_enable	Enable dual-site mode for the specified *siteid*.
site_mapping	Establish a new pin mapping between Site_0 and Site_1.
site_status	Read the state of the specified site and places it in the *state* variable.

General commands

Command	Description
devclr	Set all sources to a zero state.
devint	Reset all instruments and clear the system.
getlpterr	Get last LPTLib error since `devint` command.
getstatus	Return operating status of instrument.
insbind	Establish a cooperative relationship between two instruments.
setmode	Set operating mode.

GPIB commands

Command	Description
kibdefclr	Clear instrument on `devclr` command.
kibdefint	Clear instrument on `delay` command.
kibrcv	Read device-dependent string.
kibsnd	Send device-dependent command.
kibspl	Serial poll an instrument.
kibsplw	Synchronous serial poll an instrument.

Matrix commands

Command	Description
addcon	Add a connection.
clrcon	Disconnect all crosspoint connections.
conpin	Connect a pin or instrument terminal.
conpth	Connect pins and instruments using a specific pathway.
delcon	Remove specific matrix connections

Measure commands

Command	Description
avgX	Averages measurements of voltage, current, conductance, or capacitance.
bmeasX	Block measurement; make a series of readings as quickly as possible.
intgX	Integrates a measurement of voltage, current, conductance, or capacitance.
measX	Measure a voltage, current, conductance, or capacitance.
refctrl	Enable or disable automatic reference measurements.
setXmtr	Set the source to operate as a voltmeter or current meter.
ssmeasX	Steady state measurement (i, v).

Pulse generator commands

NOTE

These commands are only compatible with systems that have 4220-PGU pulse cards.

Command	Description
pgu_current_limit	Force a voltage or current.
pgu_delay	Set the trigger delay time.
pgu_fall	Set the fall time of the pulse.
pgu_halt	Stop all the pulse channels.
pgu_height	Set the peak-to-peak height of the pulse.
pgu_init	Initialize communication with pulse card and set pulse generator to default conditions.
pgu_load	Set the load impedance of a pulse.
pgu_mode	Set the pulse mode of the pulse generator.
pgu_offset	Set the peak-to-peak height and DC offset of the pulse.
pgu_period	Set the period of the pulse in seconds.
pgu_range	Set the voltage range of a pulse generator channel.
pgu_rise	Set the rise time of the pulse.
pgu_trig	Trigger first pulse generator unit and output waveforms.
pgu_trig_burst	Trigger a specified number of pulses.
pgu_trig_unit	Trigger a specified pulse generator unit, or units, to output waveforms.
pgu_width	Set the width of the pulse.

Range commands

Command	Description
lorangeX	Define lowest range an instrument should use during autorange operation (i, v).
rangeX	Put a measuring instrument on a specific range (c, i, v).
setauto	Re-enable autorange mode.

Scope card commands

NOTE

These commands are only compatible with systems that include 4200-SCP2HR scope cards.

Command	Description
scp_close	Disconnect communications to the scope card.
scp_detect_peaks	Return frequencies in signal amplitude order.
scp_init	Initialize the scope card to a default state.
scp_measure	Measure frequency and amplitude of the strongest signal.
scp_measure_next	Get the frequency and amplitude of next highest peak in frequency spectrum.
scp_selftest	Run an internal self-test of the scope card.
scp_setup	Set the start, stop, and step frequencies of a scan.

Spectrum analyzer commands

NOTE

These commands are only compatible with systems that include an RSA306B USB Spectrum Analyzer. In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Command	Description
rsa_close	Disconnect communications to the spectrum analyzer.
rsa_detect_peaks	Return frequencies in signal amplitude order.
rsa_init	Initialize spectrum analyzer to its default state.
rsa_measure	Measure the frequency and amplitude of the strongest signal.
rsa_measure_next	Return the frequency and amplitude of the next highest peak.
rsa_selftest	Runs the specified self-test and returns a status.
rsa_setup	Sets the start, stop, and step frequencies of a scan.

Source commands

Command	Description
forceX	Program a source instrument to output voltage or current at a specified level.
limitX	Limit an instrument to a set voltage or current other than the instrument default.
pulseX	Force voltage or current at a specified level for a specified amount of time.

Timing commands

Command	Description
adelay	Specify an array of delay points to use in a sweep.
delay	Set a user-defined delay in a test sequence (in milliseconds).
disable	Disable the timer.
enable	Initialize and start the timer.
imeast	Read the timer (immediate measure time).
rdelay	Set a user-defined delay (in seconds).

The KTE Parametric Test Subroutine (PARLib)

The Keithley Test Environment (KTE) Parametric Test Subroutine Library (PARLib) is a parameter extraction and data analysis software system. The PARLib subroutines are used to analyze data associated with parametric tests.

The following tables list the PARLib subroutines with a brief description of each subroutine. For detailed information about using the PARLib subroutines, see the "PARLib command reference" in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Bipolar subroutines

Subroutine	Description
`beta1`	Calculate DC β at specified I_E and V_{CB}
`beta2`	Calculate DC β and V_{BE} at specified I_C and V_{CE}
`beta2a`	Calculate β at V_{CB} and I_{CE} with search on I_E
`beta3a`	Calculate β at V_{CE} and I_{CE} with search on I_{BE}
`bice`	Calculate β when V_{BE} swept, at V_{CE} and V_{SUB}
`bvcbo`	Measure collector-base breakdown voltage, emitter open
`bvcbo1`	Measure collector-base breakdown voltage using LPTLib `bsweepv` subroutine
`bvceo`	Measure collector-emitter breakdown voltage, base open
`bvceo2`	Measure collector-emitter breakdown voltage using LPTLib `bsweepv` subroutine
`bvces`	Measure collector-emitter breakdown voltage
`bvebo`	Measure emitter-base breakdown voltage, collector open
`ibic1`	Measure I_{CE}, I_{BE} and calculate β at V_{CE}, V_{BE}, V_{SUB}
`icbo`	Measure collector-base leakage at V_{CB} and V_{SUB}
`iceo`	Measure collector-emitter leakage at V_{CE} and V_{SUB}
`ices`	Measure emitter-collector leakage at V_{CES} and V_{SUB}
`iebo`	Measure emitter-base leakage at V_{EB} and V_{SUB}
`rcsat`	Estimate `rcsat` when I_C and I_B swept at constant β
`re`	Estimate emitter resistance
`vbes`	Measure base-emitter voltage at specified I_E ($V_C = V_B$)

FET and JFET subroutines

Subroutine	Description
gm	Estimate MESFET transconductance at V_{DS}, V_{GS}
idss	Estimate MESFET I_{DSS} and V_{DSAT} at V_{DSS}
vp	Estimate FET pinch-off voltage for a MESFET
vp1	Estimate MESFET pinch-off at I_{DS} (I_P) and V_{DS}

Math and support subroutines

Subroutine	Description
fnddat	Search an array, return a new array
fndtrg	Determine which native mode trigger to use
kdelay	Delay, based on current and voltage values
logstp	Create an array using logarithmic steps
tdelay	Return calculated delay time

MOSFET subroutines

Subroutine	Description
bvdss	Measure drain-source breakdown voltage (V_G = 0)
bvdss1	Measure drain-source breakdown voltage using LPTLib bsweepv subroutine
deltl1	Estimate delta L MOSFET parameter
deltw1	Estimate delta W for a MOSFET
gamma1	Estimate body effect (γ)
gd	Calculate drain conductance of a MOSFET
id1	Measure drain current at specified V_{GS}, V_{DS}, and V_{BS}
idsat	Measure drain current at V_{DS}, V_{BS} ($V_D = V_G$)
idvsvg	Measure I_{DS} when V_{GS} is swept at constant V_{DS} and V_{BS}
isubmx	Find peak substrate current at V_{DS},V_{BS}
vg2	Measure gate-source voltage at I_{DS}, V_{DS}, V_{BS}
vgsat	Measure V_{GSAT} at specified I_{DS} ($V_{GS} = V_D$)
vt14	Estimate V_T using two-point technique
vtati	Find V_T to produce specified I_{DS}
vtext	Extrapolate gate-source threshold voltage
vtext2	Estimate V_T using modified vtext subroutine method
vtext3	Calculate V_T using max slope method

Resistors, diodes, capacitors, and special structure subroutines

Subroutine	Description
bkdn	Measure breakdown voltage (force I, measure V)
cap	Measure two-terminal capacitance
fimv	Force current and measure voltage on device with four high pins and four ground pins
fvmi	Force voltage and measure current on device with four input pins and four ground pins
leak	Measure leakage current at specified voltage
res	2-terminal resistance (force I, measure V)
res2	2-terminal resistance with voltage limit
res4	4-terminal resistance (force I, measure V)
resv	2-terminal resistance (force V, measure I)
rvdp	4-terminal van der Pauw measurement
tox	Calculate oxide thickness from capacitance
vf	Measure the forward junction voltage of a diode

The High-Voltage Library (HVLib)

The Keithley High-Voltage Library (HVLib) is a set of commands you can use to make measurements on an S540 Power Semiconductor Test System using the Keithley Test Environment (KTE) software.

You can use these commands in two- and three-terminal measurement applications to measure capacitance-voltage (C-V), calculate compensation constants, do open, load, and short compensation, and do breakdown voltage tests.

For detailed information about using the HVLib commands, see High-voltage C-V measurements (on page 9-1) in this manual and the "HVLib command reference" in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

High-Voltage Library (HVLib) commands

The following table contains the High-Voltage Library (HVLib) commands and a brief description of each. For detailed descriptions of each of the commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`gate_charge`	Measures the gate charge required to switch on the power transistor.
`hv_bvsweep`	Does a breakdown voltage sweep.
`hvcv_3term`	Measures output capacitance, input capacitance, or short-circuit reverse transfer capacitance of three-terminal devices.
`hvcv_3term_basic`	Makes basic measurements of output capacitance, input capacitance, or short-circuit reverse transfer capacitance of three-terminal devices.
`hvcv_comp`	Does complex mathematical calculations to implement specified impedance compensation models.
`hvcv_genCompData`	Generates correction factors for system-level high-voltage capacitance-voltage (C-V) compensation.
`hvcv_genCompFreq`	Generates compensation factors for system-level capacitance compensation for a single, specified frequency.
`hvcv_getData`	Gets compensated capacitance and compensated conductance data from the data pool.
`hvcv_intgcg`	Measures capacitance and does system-level ShortOpenLoad compensation on the high-voltage capacitance meter (CMTR).
`hvcv_measure`	Measures and stores compensated capacitance and compensated conductance values.
`hvcv_storeData`	Stores compensated capacitance and compensated conductance data in the data pool.
`hvcv_sweep`	Does a high-voltage C-V sweep.
`hvcv_sweep_basic`	Does a basic high-voltage C-V sweep.
`hvcv_test`	Makes a high-voltage C-V measurement at a single frequency.
`hvcv_test_basic`	Makes a basic high-voltage C-V measurement at a single frequency.

Section 6

Software

In this section:

Introduction

This section describes the KTE software and software tools that you can use to create and execute test programs:

- Creating test plans: Discusses the tools used to create test plan files.
- Adaptive testing (on page 6-95): Describes the optional component of the Keithley Test Environment (KTE) that enables the test plan to change for each wafer being tested.
- Test execution: Discusses the tools that control the selection and execution of cassette plans.
- Data analysis: Describes the tools used to gather and analyze results data provided by an executed test program.
- System administration: Details the setup of the KTE software, file management, system configuration, and ethernet specifications.
- Data logging: (on page 6-174) Describes control and handling of test data and the routines used to organize and save parametric test data into ASCII data files.
- Keithley User Interface Library: Describes the different subroutines that support the creation of sophisticated operator interface dialogs for test programs.
- KTE file formats (on page 6-201): Provides file structure examples for all of the files created using the KTE software.

- Data pool (on page 6-228): Describes the data pool which is used to hold global data while the Keithley Test Execution Engine (KTXE) is running.
- User access points: Provides descriptions and examples on how to use User Access Points.

Keithley Test Environment system software

The S540 parametric test system includes integrated Keithley Test Environment (KTE) software that provides fast, flexible test plan development and high-speed test execution for process control monitoring, process reliability monitoring, and device characterization applications.

NOTE

Earlier versions (V2.X and earlier) of the KTE software created compiled, uneditable, executable files. Later versions of the software create a much more flexible, editable test plan.

Features include:

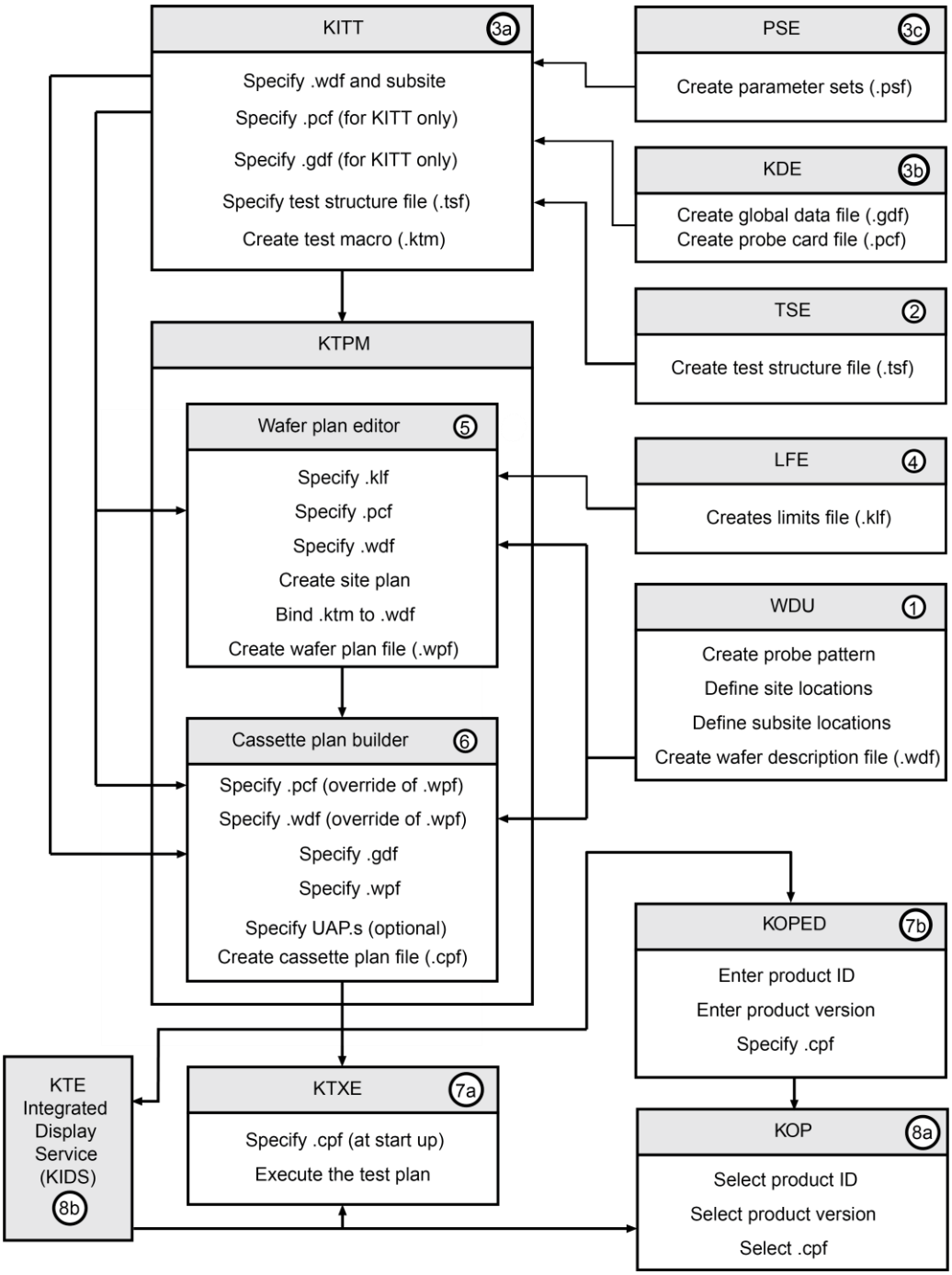
- Test plans that you can edit for your specific needs
- User Access Points (UAPs) that allow you to extend the functionality of the test execution engine for conditional test sequencing and customization of the flow of system operation
- Test execution that is data driven; test plan execution is derived from data files read by the test execution engine
- The Wafer Description Utility (WDU) describes the wafer and specifies the test sites and subsites
- The Limits File Editor (LFE) specifies the test result limits
- The Keithley User Library Tool (KULT) allows you to create user libraries that are accessible in the Keithley Interactive Test Tool (KITT)
- KITT allows you to create a test macro that will be run on a selected wafer
- The Keithley Data Editor (KDE) gives you access to all probe card files (`.pcf`), global data files (`.gdf`) and predefined identifiers (PDI) from one centralized location
- The Test Structure File Editor (TSE) allows you to create a set of test parameters for each different device to be tested
- The Parameter Set Editor (PSE) allows you to edit and add parameter set data for a specific routine
- The ability to create probe card files (`.pcf`) that specify pad name to tester pin assignments
- The ability to create global data files (`.gdf`) that assign values to test data variables and can be accessed by multiple test plans
- The ability to create wafer plans (`.wpf`) that specify the wafer description, site, and subsite plans
- The ability to create limits files that control the wafer testing depending on the test results

- The ability to create cassette test plans (`.cpf`) that tie all of the test data together into a coherent executable test plan
- The ability to use math expressions in test macros and conditional execution of test modules.
- Compatible with earlier systems
- Runs on a standard industrial PC with a Linux® operating system

KTE process overview

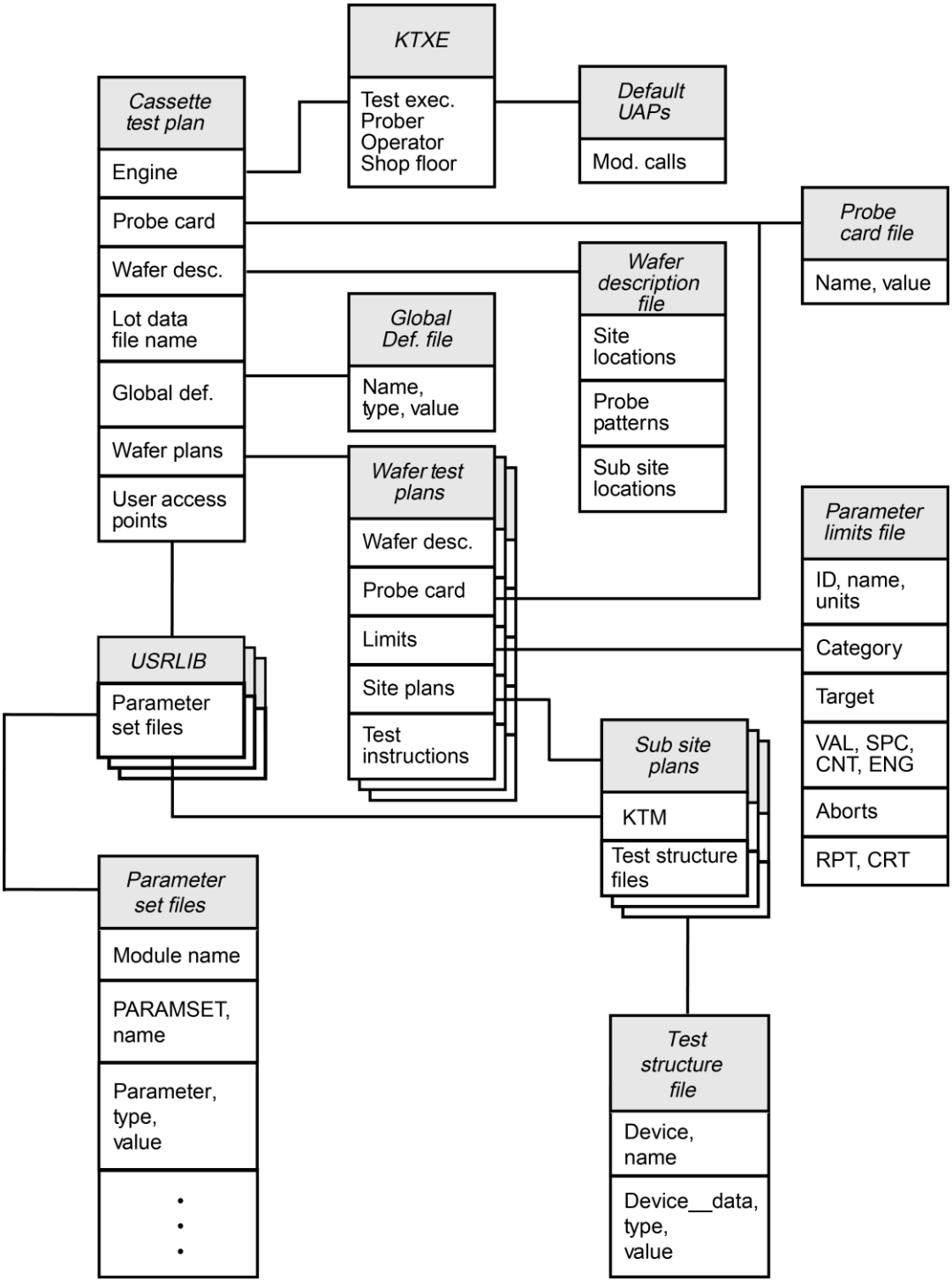
The S540 system uses the Keithley Test Equipment (KTE) Software platform. The following figure shows a diagram of how Keithley Test Environment (KTE) software is used to create and execute a cassette plan.

Figure 17: Cassette plan creation flow diagram



All of the data files created using KTE play important roles in creating a coherent test plan. The following figure shows a diagram of the data in each file and the placement of each file when a test plan is created.

Figure 18: Data file hierarchy



During test plan creation, it is possible to specify the probe card file and the wafer description file in more than one place. If this occurs, the file specified in the Cassette Test Plan takes precedence over the file specified in the Wafer Test Plan.

KTE software tools

The Keithley Test Environment (KTE) software includes tools that make it easier for you to create and execute the test programs for the S540 test system. The KTE software tools are described in the following table.

KTE software tools

Name	Description
Wafer Description Utility (WDU) (on page 6-15)	Capture information to describe the size of a wafer, its orientation, sites, and subsites.
Test Structure File Editor (TSE) (on page 6-37)	Create test structure files that contain device-specific parameters.
Keithley Interactive Test Tool (KITT) (on page 6-42)	Create test macros (including the data used), and interactively debug the macros on the tester
Keithley Data Editor (KDE)	A function in KITT; create and modify global data files and probe card files.
Parameter Set Editor (PSE) (on page 6-57)	Edit and add parameters for a library module routine.
Keithley User Library Tool (KULT) (on page 6-60)	Create test modules and user libraries that can be used in KITT and as user access points (UAPs) to control the testing process.
Limits File Editor (LFE) (on page 6-78)	Capture information describing the criteria for judging the measured parameters identified in KITT.
Keithley Test Plan Manager (KTPM) (on page 6-83)	Create wafer test plans, cassette plans, and cassette plan documentation.
Keithley Operator Interface (KOP)	Operator interface; select and execute a cassette plan.
Keithley Operator Interface Editor (KOPED) (on page 6-106)	Create the operator's cassette plan selection tree that is presented to the operator in KOP.
Keithley Test Execution Engine (KTXE) (on page 6-108)	Execute the cassette plans created using KTPM; specified in the cassette plan.
KTE Integrated Display Service (KIDS)	Select and execute recipes, display status, and the 300mm automation option (if installed).
Keithley Summary Utility (KSU) (on page 6-133)	View the results and compare them against the parameter limits entered in the Limits File Editor.
Keithley Curve Analysis Tool (KCAT) (on page 6-146)	Create a graphical representation of array results data generated from a test run using KITT.
KDFtoKCS File Conversion Utility (on page 6-150)	Create Cornerstone format files (`.kcs`) from KDF files.

KTE support utilities

The following topics describe some utilities that you can use.

Random pattern generation

The random pattern generation (`rand_pat`) utility provides the ability to read in a `.wdf` file containing zones (patterns) and create a new `.wdf` file containing randomly generated patterns covering the given zones. This function may be used as a stand-alone utility.

Syntax of command:

```
rand_pat infile [-s sites/wafer] [-w wafers/cassette] [-o outfile]
```

Where:

`infile` = Input `.wdf` file with define zones (patterns)

`[sites/wafer]` = Number of sites per wafer

`[wafers/cassette]` = Number of wafers per cassette

`[outfile]` = Output `.wdf` file

Each time the routine is called, a new random seed is generated based on the date and time returned by the system. This guarantees full randomness over the set of all possible combinations of sites and sectors. The output file will contain patterns named `pattern_n`, where `n` is the number of the pattern. For example, if the output file contains two patterns, the patterns would be named `pattern_1` and `pattern_2`.

Note that if neither the `-s` or `-w` option is used, the routine will set the number of sites per wafer equal to the number of zones, and then generate a natural set of patterns without any retesting. If the number of wafers is a multiple of the number of zones, and each zone has the same number of sites, then even coverage is uniformly achieved.

At this point, the utility allows you to specify a number of sites per wafer different from the number of patterns. It also allows you to generate a number of patterns different from the natural number. If fewer patterns are specified, there may be untested sites; if more are specified, then there will be redundant coverage. Note that if more than the natural set is generated, the random seed is reinitialized, and all following patterns will again be truly random.

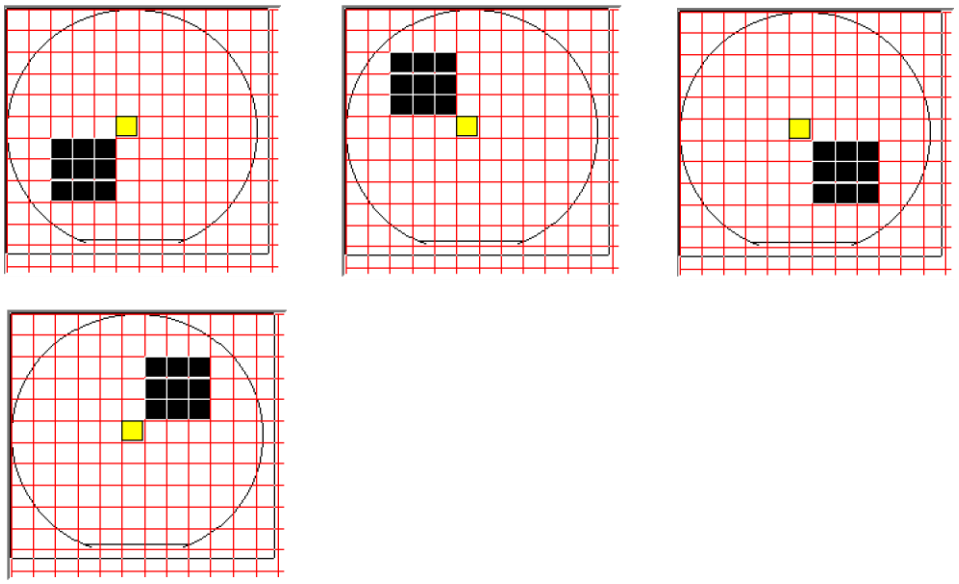
Example:

Upon executing

```
> rand_pat mywdf.wdf -o randwdf.wdf
```

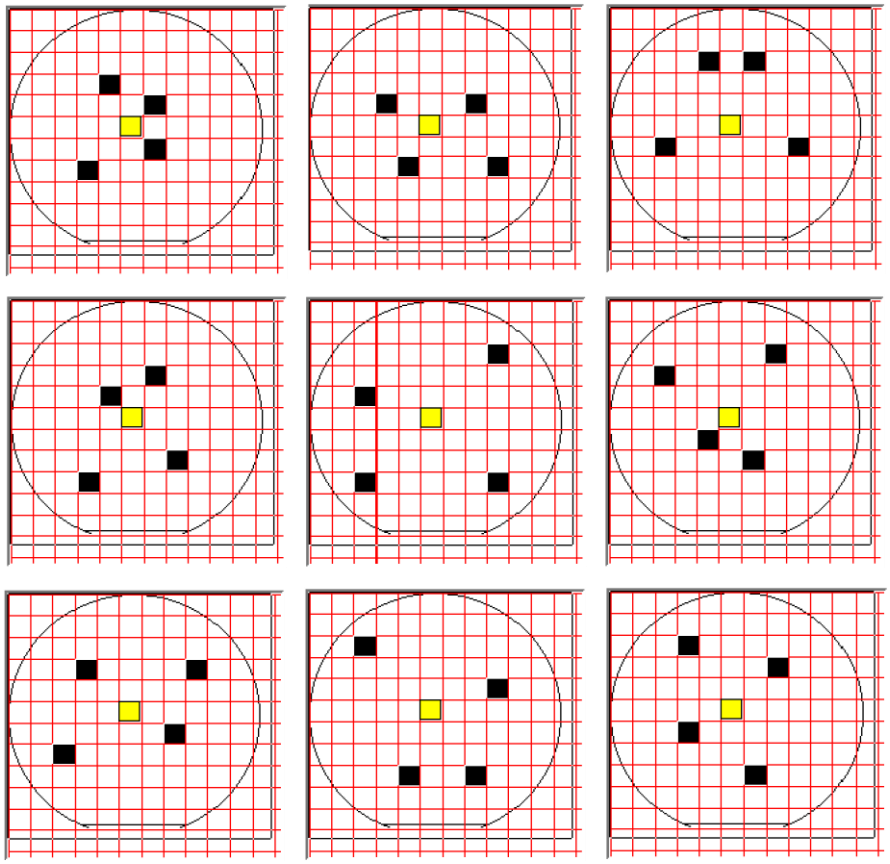
Where the input file zones have the following form:

Figure 19: Random pattern utility input file zones



The output patterns look like:

Figure 20: Random pattern utility output patterns



File filtering

All Keithley Test Environment (KTE) tools use a standard window to open and save files. Directories are displayed with a folder icon before the name, and read-only files are identified by a lock icon before the filename. A Choose Filter drop-down menu allows for the selection of predefined filter patterns. Files may be filtered by entering a new pattern in the Filter field and clicking the **Filter** button.

The predefined filter list provides a simple way to filter files by picking from a list of custom filters. These filters are listed by a simple name for easy recognition. The filter list can be customized using a plain text editor. The filter list is located in the file `$KIHOME/filters.ini` file. The format of the file is as follows:

```
[filter nice name]filterPattern
```

Where `[filter nice name]` is the name that will appear in the window drop-down listing. `filterPattern` is a simple expression filter used for pattern matching. The filter pattern supports only two wildcard characters: `?` to indicate any single character, and `*` to indicate any substring of characters. The filter name must be enclosed by the brackets.

For example:

```
[Project 1 Files]project_1_*
```

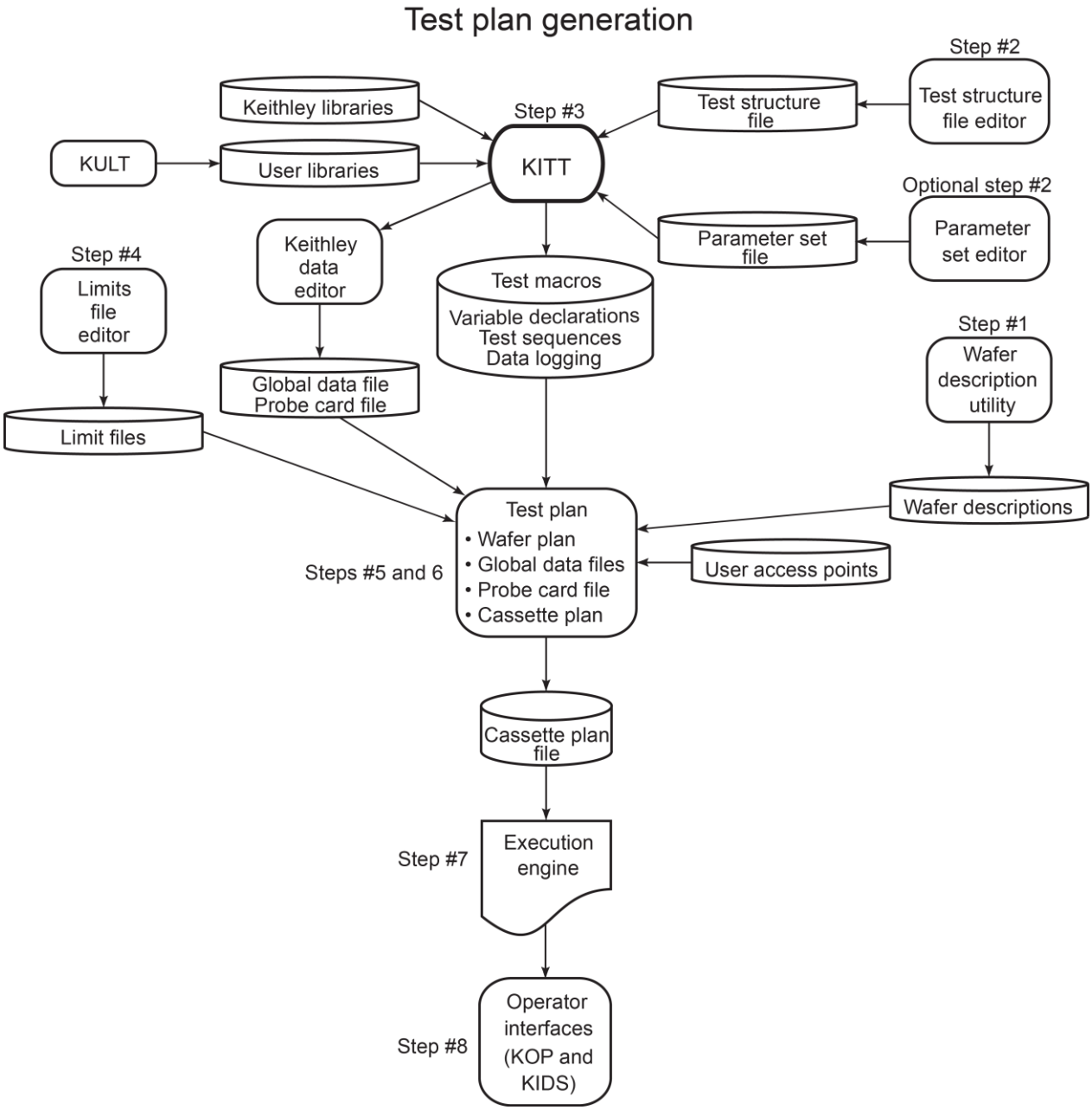
This filter would be displayed in the open and save windows as Project 1 Files, and would display all filenames that started with the characters `project_1_`.

Comments may be added to the `filters.ini` file by preceding them with the `#` character. The comment will not appear in the Choose Filter drop down box. The period character (`.`) may not be used as part of the filter pattern.

Building and executing tests

Production-quality test plans are built and executed using the Keithley Test Plan Manager (KTPM). KTPM allows you to create sophisticated test plans without having to edit, compile, and link C language source files. Instead, you create your test plans by selecting from prebuilt program elements and allow KTPM to bind them together to produce the final cassette plan. The following figure shows the test plan creation process.

Figure 21: Test plan generation block diagram



The Keithley Test Execution Engine (KTXE) accesses the Cassette Test Plan, which specifies the:

- Execution engine
- Probe card file
- Wafer description file
- Lot data filename
- Global data file
- Wafer test plans

The cassette plan then accesses this data. If the probe card file and wafer description file have already been specified by the cassette plan, they are ignored if specified in the wafer plan file.

For more detailed information about KPTM, see Keithley Test Plan Manager (KTPM) (on page 6-83). For more detailed information about KTXE, see Keithley Test Execution Engine (KTXE) (on page 6-108).

Creating test plans

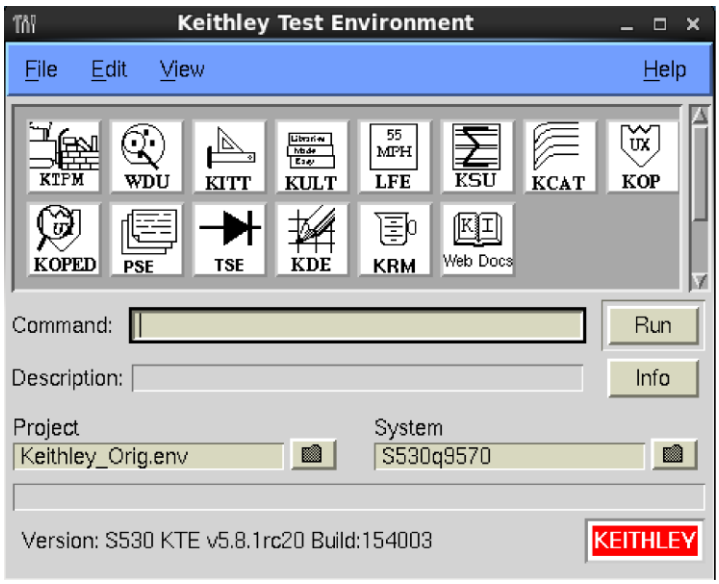
This section describes the KTE user interface and the tools required to create the files used when building a test plan, including:

- Keithley Tool Pallet (on page 6-13): Overview of the KTE User interface. Using the Keithley Tool Pallet (KTP) in a point-and-click interface for the Keithley Test Environment (KTE). The tool palettes for the S540 system, shown in the following figure, contain an icon for each of the test plan development and utility functions. Refer to System administration for further details about Keithley Tool Pallet.
- Wafer Description Utility (on page 6-15): Using the Wafer Description Utility (WDU) to create a wafer description file that describes the wafer used during the testing process.
- Test Structure File Editor (on page 6-37): Using the Test Structure File Editor (TSE) to create parameters for a specified device located at a specific wafer subsite.
- Keithley Interactive Test Tool (on page 6-42): Using the Keithley Interactive Test Tool (KITT) to create a test macro containing the tester commands.
- Parameter Set Editor (on page 6-56): Using the Parameter Set Editor (PSE) to modify parameter set data assigned to a library module.
- Keithley User Library Tool (on page 6-60): Using the Keithley User Library Tool (KULT) to create a user library that contains test modules that can be used when creating a test macro.
- Limits File Editor (on page 6-78): Using the Limits File Editor (LFE) to create a limits file containing parameter limits that are compared to the results acquired during testing.
- Keithley Test Plan Manager (on page 6-83): Using the Keithley Test Plan Manager (KTPM) to create a cassette plan file (`.cpf`) that can be executed by the Keithley Test Execution Engine (KTXE).

Keithley tool palette

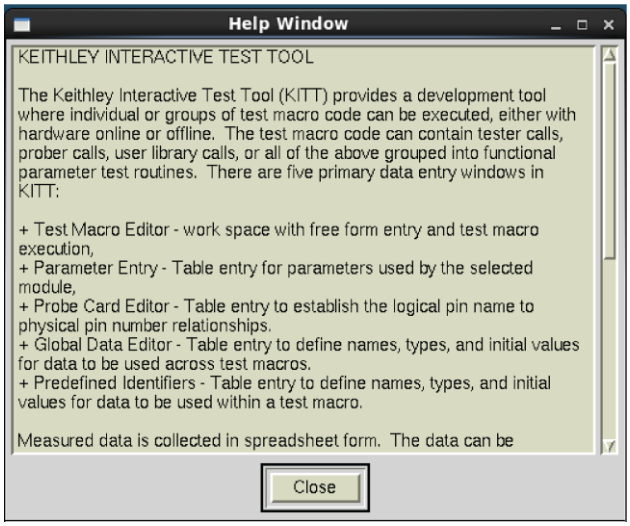
The tool palette provides a point-and-click interface into the Keithley Test Environment (KTE) development environment. The tool palette for the S540 system, shown in the following figure, contains an icon for each of the test plan development and utility functions.

Figure 22: Keithley tool palette



After clicking an icon, the command related to that icon is displayed in the Command field and a brief description of the selected tool is displayed in the Description field. You can access more information about the selected command by clicking the Info button located in the lower right corner of the tool palette. This opens the Help window shown in the following figure.

Figure 23: Tool window



To start the currently selected tool, click the **Run** button in the lower right corner of the tool palette or double-click the icon of the software tool you want to initiate. The status of the program selected is displayed directly below the Description field.

Menu bar

The menu bar is located across the top of the tool palette and provides access to the following system-related functions.

File menu

- **Exit:** Closes the Keithley tool palette.

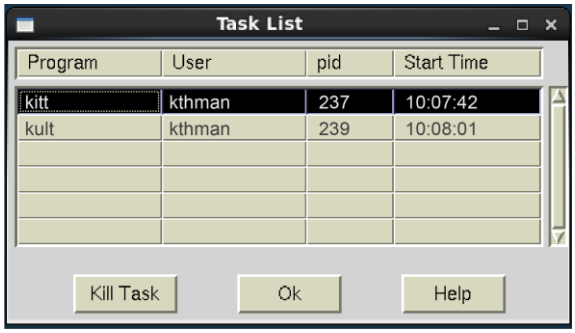
Edit menu

- **Edit tool.tpi:** Allows you to modify the tool palette. The `tool.tpi` file controls the icons displayed by each of the View menu selections.
- **Reload tool.tpi:** Reinitializes the program using the modified `tool.tpi` file. The `tool.tpi` file must be reloaded after any changes have been made for modifications to take effect.

View menu

- **Keithley tools:** Displays Keithley Development Tools icons.
- **Tester Tools:** Displays the Tester Tools icons.
- **Linux® tools:** Displays the System Tools icons.
- **Linux administration:** Displays the Administrator Tools icons.
- **User programs:** Displays the User programs icons.
- **All:** Displays all of the icons listed above.
- **Task list:** Displays a dialog box containing presently running programs, as shown in the following figure, and lets you stop individual program execution.

Figure 24: Task list window



- **Icon only view:** Removes everything from the tool palette except the icons and opens the task list. The right mouse button accesses a dialog box that lets you toggle between the icon-only view and the full view of the tool palette.

Help menu

- **KTP documentation:** Lists version information about the Keithley Test Environment (KTE) software.
- **About:** Opens the help window related to the software tool selected (the same as clicking the Info button).

Project field

The project field lets you select a specific project environment that describes the location of the Keithley Test Environment (KTE) directory tree to use during testing.

System field

The system field lets you specify the test system to which your workstation should connect.

Wafer Description Utility (WDU)

The Wafer Description Utility (WDU) captures the size and location information that describes where the test structures that will be probed are when this wafer description file is used. This information is entered several different windows within the WDU:

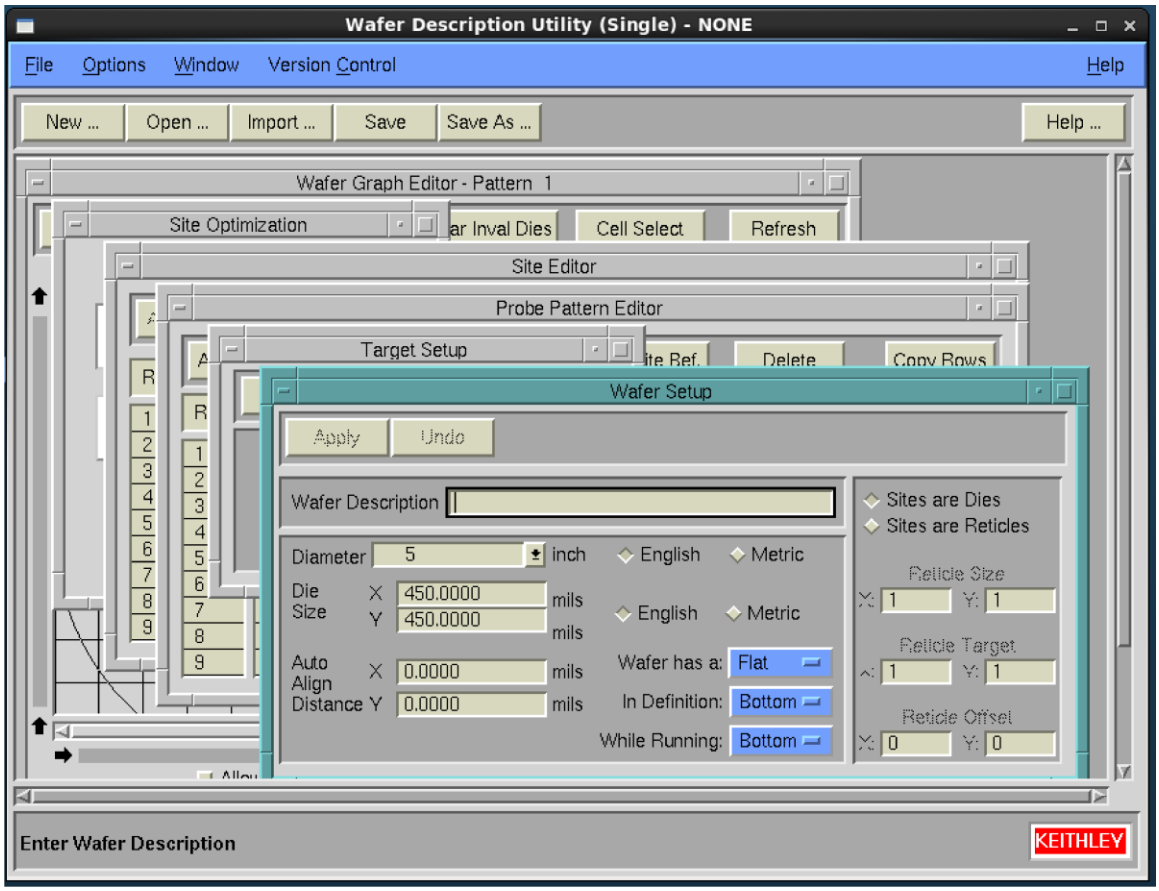
- Wafer Setup
- Target Setup
- Probe Pattern Editor
- Site Editor
- Site Optimization
- Wafer Graph Editor
- User Defined Values

The resulting file is saved in an easy-to-read ASCII format.

WDU main window

The Wafer Description Utility (WDU) main window appears when WDU is initiated. A description of the main window components follows.

Figure 25: WDU main window



Title bar: The title bar displays the filename of the current wafer description file, and the read-write status of the file.

Menu bar: The menu bar contains selections used to modify the information in the main window work area.

Control buttons: The control buttons (New, Open, Import, Save, Save As, and Help) give you control over the information in the main window work area.

Main window work area: The main window work area contains all the windows used to enter the data for each wafer description file.

WDU File menu

Selecting **File** produces the menu shown in the following figure.

Figure 26: WDU File menu

New: Initializes a new `.wdf` file as either single or multiple project.

Open: Opens an existing `.wdf` file.

Import: Imports or converts a file to a `.wdf` file. The import script must be specified in the `wdu.ini` file. See the Wafer Description Utility (WDU) online help for more details.

Save: Writes the current `.wdf` to the path and filename shown in the title bar.

Save As: Writes the current `.wdf` file to a user-specified path and filename.

Delete: Opens a window allowing you to delete a file.

Print: Allows you to print data from the Site Editor, Wafer Graph Editor, Target Setup, Wafer Setup, Probe Pattern Editor, and Site Optimization screens.

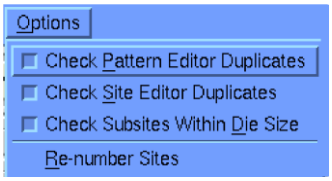
Print Setup: Lets you set up the printed page parameters.

Exit: Exits the WDU.

WDU Options menu

Selecting **Options** from the menu bar produces the menu in the figure below.

Figure 27: WDU Options menu



Check Pattern Editor Duplicates: Checks for duplicate probe pattern IDs, cross-checks the site IDs in the Probe Pattern Editor against those in the Site Editor, and checks for duplicate site locations within a pattern, all when saving the file.

Check Site Pattern Duplicates: Checks for duplicate subsite IDs and for duplicate site IDs when saving the file.

Check Subsites Within Die Size: Verifies that all subsite coordinates are within the die size when saving the file. The Wafer Description Utility (WDU) uses the absolute values of subsite coordinates in its comparison against the die size.

Re-number Sites: Renumbers site IDs. Any existing site IDs are replaced with the default names, `Site_N`, where *N* is an increasing number. This allows you to reorder site names and numbers. You are prompted before any action is taken.

WDU Window menu

Selecting **Window** from the menu bar produces the following menu.

Figure 28: WDU Window menu

Cascade: Arranges all windows so the title bars of each can be seen in the main window.

Tile: Arranges all the windows so they are viewed in the main window work area.

Arrange Icons: Arranges the icons for all the closed windows along the bottom of the main window work area.

WDU Help menu

Selecting **Help** produces a menu that accesses the following items:

WDU Documentation: Provides help for the Wafer Description Utility (WDU).

About: Displays the installed WDU version number.

Control buttons

Clicking a control button performs one of the following procedures:

New: Allows the selection of Single or Multi-project mode, and initializes the Wafer Description Utility (WDU) for the entry of a new `.wdf` file.

Open: Opens an existing `.wdf` file.

Import: Imports or converts a file to a `.wdf` file. Import script must be specified in the `wdu.ini` file. See the WDU online help for more details.

Save: Writes the current `.wdf` file to the path and filename shown in the title bar.

Save As: Writes the current `.wdf` file to a user-specified path and filename.

Help: Provides online help information about the WDU.

Main window work area

The main window work area is directly below the control buttons. This area contains all of the windows used to enter data for each wafer description file:

Wafer Setup: Lets you enter data describing the wafer (For example, wafer size and notch location).

Target Setup: Lets you position the wafer target and change the wafer orientation.

Probe Pattern Editor: Lets you add or remove wafer locations from probe patterns (group of site locations on the wafer).

Site Editor: Lets you add or remove subsites from within sites.

Site Optimization: Lets you select the site probing order.

Wafer Graph Editor: Graphically shows die locations that are selected (single project only) and allows cursor selection of additional die locations.

NOTE

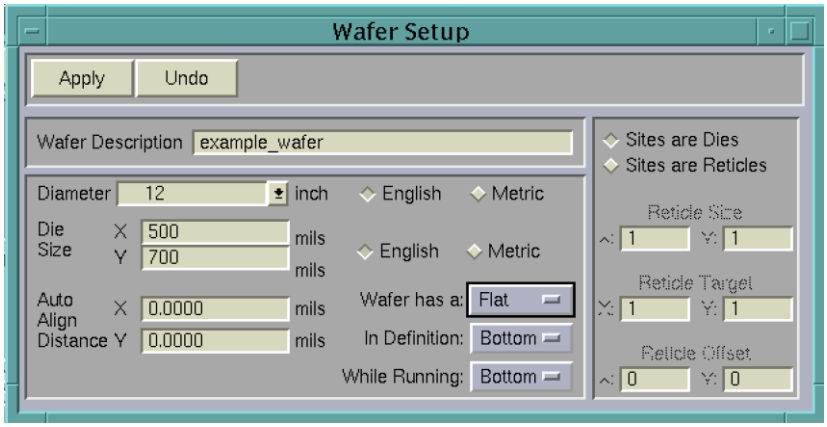
The center mouse button can be used to select any of these windows when the cursor is in the main window work area.

The functions of each window are discussed in detail in the subsections that follow.

Wafer Setup window

The Wafer Setup window, shown in the following figure, is used to enter all of the wafer size, die size, site definition, and autoalign distance data.

Figure 29: Wafer Setup window



Control buttons

Apply: Applies the wafer setup information to all other windows.

Undo: Resets the field values to the values present at the time of the previous Apply.

Wafer setup data fields

Wafer Description: Lets you enter a short description of the `.wdf` file.

Diameter: Lets you set the diameter of the wafer (inches or millimeters) from a default list of sizes stored in the `wdu.ini` file by clicking the down arrow and making the selection.

Die Size: Lets you enter the X and Y die size (mils or millimeters). These fields are not used for multiproject wafers.

Auto Align Distance: Lets you enter the X,Y position for a prober move that could be used to position the wafer for manual adjustments. User Access Point (UAP) code must be written and referenced in the cassette plan to use this data at run time.

English-Metric selector buttons

The top set of buttons lets you select English (inches) or Metric (millimeters) as the measurement setting for the wafer diameter. The bottom set of buttons lets you select English (mils) or Metric (millimeters) as the measurement setting for the rest of the wafer data.

Wafer orientation buttons

Pop-up windows are activated when positioning the cursor on the following selection rectangles and holding the left mouse button down.

Button preceded by "Wafer has a" (flat or notch button): Select flat or notch.

Orientation buttons: Specify the flat or notch position to be on the left, right, bottom, or top of the wafer in the following two categories:

- **"In Definition" (Bottom, Top, Left, or Right button):** Specifies the normal flat or notch orientation.
- **"While Running" (Bottom, Top, Left, or Right button):** Specifies the orientation to which the flat or notch may be rotated when in the prober (for example, to probe the "streets" in a particular direction).

Site definition area

Sites are Dies button: Specifies that each site is an individual die, in contrast to a group of dies, as defined by a reticle. Accordingly, the reticle dimension and coordinate fields in the site definition area are disabled.

Sites are Reticles button: Specifies that each site is a group of dies, as defined by a reticle. The sets of fields described below specify the size and coordinates of the reticle.

- **Reticle Size field:** When Sites are Reticles is selected, Reticle Size specifies the X,Y dimensions of a reticle in terms of the number of sites in each direction. For example, if X: is 4 and Y: is 4, the reticle is 4 sites wide and 4 sites deep and contains 16 dies.
- **Reticle Target field:** When Sites are Reticles is selected, defines the relative X,Y coordinates of the reticle's target die in terms of the number of sites in each direction. For example, if Reticle Target coordinates are 2,2 and Reticle Offset coordinates are 0,0, the target die is located in the second column from the left and the second row from the bottom.
- **Reticle Offset field:** When Sites are Reticles is selected, Reticle Offset defines the relative X,Y location on the reticle from which internal coordinates are measured. For example, if Reticle Offset is 0, 0, the internal coordinates are measured from origin of the reticle (lower left corner).

Target Setup window

The Target Setup window shown in the following figure, sets the X,Y coordinates of the target site (die or reticle) and the direction for increases in the X,Y coordinate values.

Figure 30: Target Setup window

Control buttons

Recalc.: Saves target information and adjusts the coordinate values of all specified sites so that the relative position of the target site is maintained.

Apply: Saves target information only. Sites specified will shift relative to the new target.

Undo: Resets the field values to the values present at the time of the previous Apply.

X and Y data fields

These fields let you specify the target X and Y coordinate values in site (die or reticle) units for single project wafers, or in mils and millimeters for multiproject wafers (multiproject wafers have more than one repeating reticle pattern).

X dir and Y dir arrow buttons

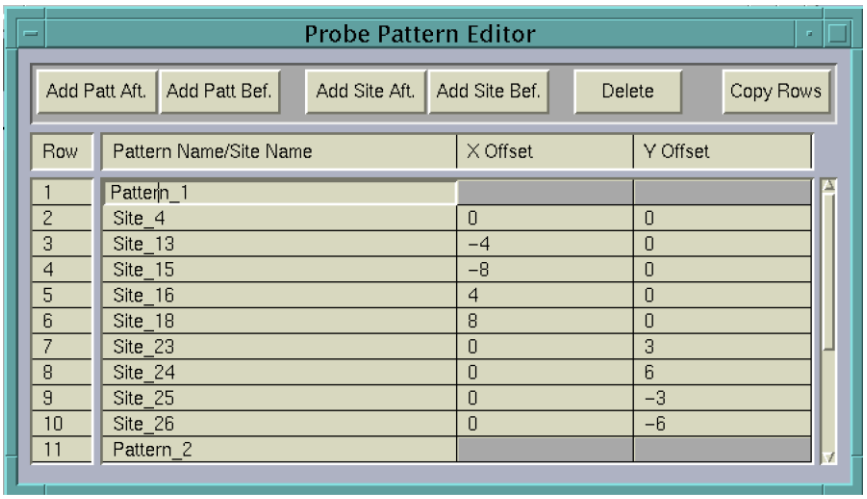
These buttons let you set the direction in which the X and Y position values increase.

Probe Pattern Editor window

The Probe Pattern Editor window, shown in the following figure, is used to specify site probe patterns during testing. These site probe patterns are paired with tests in the Keithley Test Plan Manager (KTPM) tool's Wafer Plan Editor window. This allows a simple method for specifying different tests at different sites.

All the tests for a specific site from multiple patterns are combined by the execution engine so that each site is visited by the test probe one time only. Default site names are given to each site in a pattern. You can change this default name by clicking the site name and entering a different name for the site.

Figure 31: Probe Pattern Editor window



Right-clicking the mouse button in the Pattern Name field brings up a menu that lets you move quickly through the different patterns you have created. The commands that are accessed through this menu are:

Goto Next Pattern: Jumps to the next pattern in the list.

Goto Prev. Pattern: Jumps to the previous pattern on the list.

Find Pattern: Opens a dialog box that lets you enter the name of the pattern you want to change.

Find Site: Opens a dialog box that lets you enter the name of the site you want to change.

Find Site Location: Opens a dialog box that lets you enter the coordinates of the site location you want to change.

Reset Pattern/Site List: Removes all patterns and sites from the Pattern Editor window.

Control buttons

The control buttons along the top of the Probe Pattern Editor window function for both single and multiproject files:

Add Patt Aft.: Adds a pattern name field to the pattern listing after the currently selected field.

Add Patt Bef.: Adds a pattern name field to the pattern listing before the currently selected field.

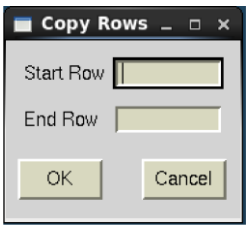
Add Site Aft.: Adds a site name field to the site listing after the currently selected field. Site names are not necessary for single project wafers because all sites are identical.

Add Site Bef.: Adds a site name field to the site listing before the currently selected field. Site names are not necessary for single project wafers because all sites are identical.

Delete: Deletes the currently selected field.

Copy Rows: Copies the rows indicated in the Copy Rows window, shown in the following figure, after the row where the cursor is currently positioned.

Figure 32: Copy Rows window



Pattern Name/Site Name data field

The information in this field depends on the project type selected with the New button. A pattern name must be entered before any sites can be added. The default name `Pattern_1` is entered by the Wafer Description Utility (WDU) when a new `.wdf` file is started.

Single project: Site names are not necessary, but can be entered for a single project wafer, because all the dies on the wafer are the same. A `.wdf` file can contain many probe patterns.

Multiproject: Each site within a pattern must be given a name because a multiproject pattern is not restricted to a single, repeating reticle frame. Many probe patterns are also supported in a multiproject wafer file.

X and Y offset fields

The information in these fields differs depending on the project type selected.

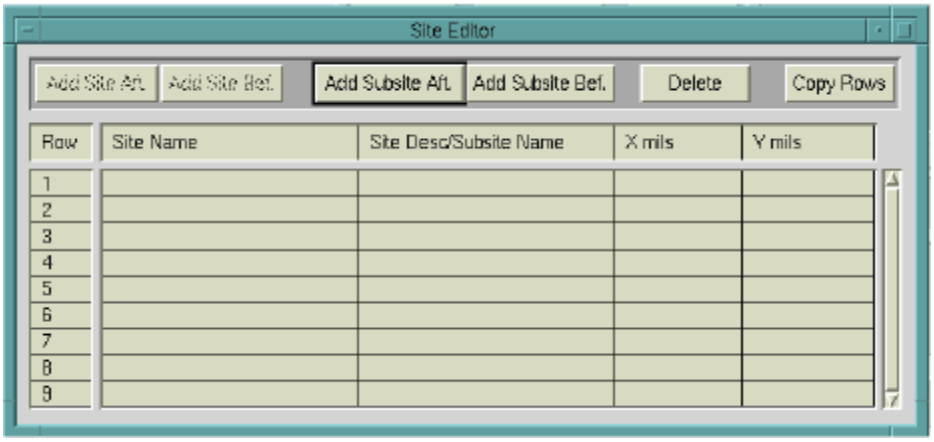
Single project: The X,Y offset values are based on site (die or reticle) coordinates. You specify the coordinate values through the Target Setup window.

Multiproject: The X,Y values provided are position information in millimeters or mils. These values are absolute position information. The wafer alignment can be used to put the wafer in the correct initial position.

Site Editor window

The Site Editor window, shown in the following figure, is used to specify the different sites and their subsites. The information required depends on the project type selected.

Figure 33: Site Editor window

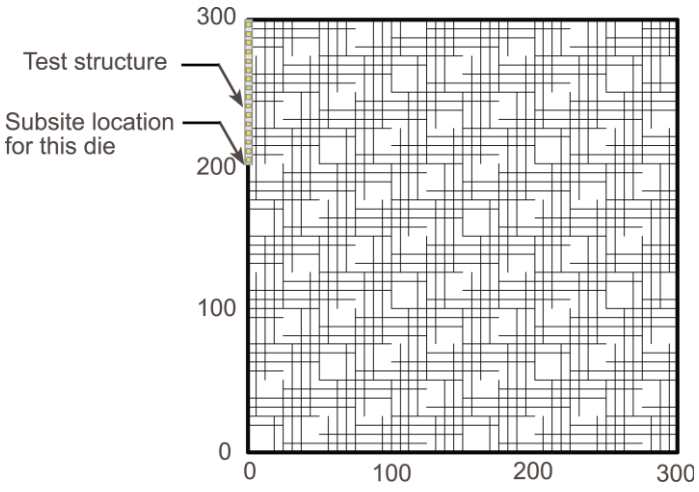


Single-project wafers have only one site type; no site information is necessary – only subsite information. Multiproject wafers can have more than one site type. Therefore, both the site's description and subsite locations must be provided.

To specify subsite X,Y coordinates:

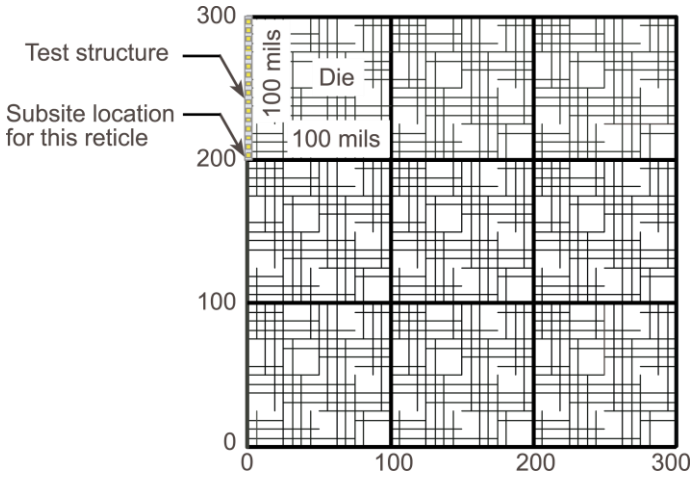
- For a site that is defined as a die, specify subsite coordinates relative to the dimensional origin of the die (typically the lower left corner of the die). See the following figure.

Figure 34: Subsite coordinates 0,200 for a site that is defined as a single 300 mil x 300 mil die



- For a site that is defined as a reticle, specify subsite coordinates relative to the dimensional origin of the reticle (typically the lower left corner of the reticle), even though the subsite may be a test structure on one of the dies that compose the reticle. See the following figure.

Figure 35: Subsite coordinates 0,200 for a site defined as a reticle with nine 100 mil x 100 mil



Right-clicking the mouse button in the Site Name field brings up a menu that lets you move quickly through the different patterns you have created. The commands that are accessed through this menu are:

Find Site: Opens a dialog box that lets you enter the name of the site you want to change.

Find Subsite: Opens a dialog box that lets you enter the name of the subsite you want to change.

Find Subsite Location: Opens a dialog box that lets you enter the coordinates of the subsite location you want to change.

Reset Site/Subsite List: Removes all sites and subsites from the Editor window.

Open .tsf: Opens the corresponding test structure file in the Test Structure Editor (TSE). If the file does not exist, you are prompted to create the file or cancel.

Control buttons

The control buttons along the top of the Site Editor window function as follows:

Add Site Aft.: Used for multiproject wafers to provide site information after the currently selected field for a site name listed in the Probe Pattern Editor.

Add Site Bef.: Used for multiproject wafers to provide site information before the currently selected field for a site name listed in the Probe Pattern Editor.

Add Subsite Aft.: Adds a subsite after the currently selected field.

Add Subsite Bef.: Adds a subsite before the currently selected field.

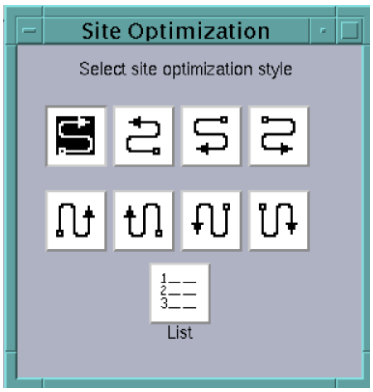
Delete: Deletes the currently selected field.

Copy Rows: Copies the rows indicated in the Copy Rows window after the row where the cursor is currently positioned.

Site Optimization window

The Site Optimization window, shown in the following figure, is used to specify the order in which each of the test sites on the wafer is tested.

Figure 36: Site Optimization window



You may choose from eight different patterns and the List selection. When a pattern is selected, the tester starts at the site closest to the chosen location and then proceeds to all selected sites on the wafer, following the selected serpentine pattern. When the List selection is chosen, the system tests each of the sites in the order they are listed in the Probe Pattern Editor window.

Wafer Graph Editor window

The Wafer Graph Editor window, shown in the following figures, works with single-project `.wdf` files only. This window displays a single probe pattern at a time or all probe patterns contained in a `.wdf` file the Show All Patterns button is above the wafer display). The left mouse button can be used to select additional site locations to be included in a probe pattern.

Figure 37: Wafer Graph Editor when "Sites are Dies" is selected in

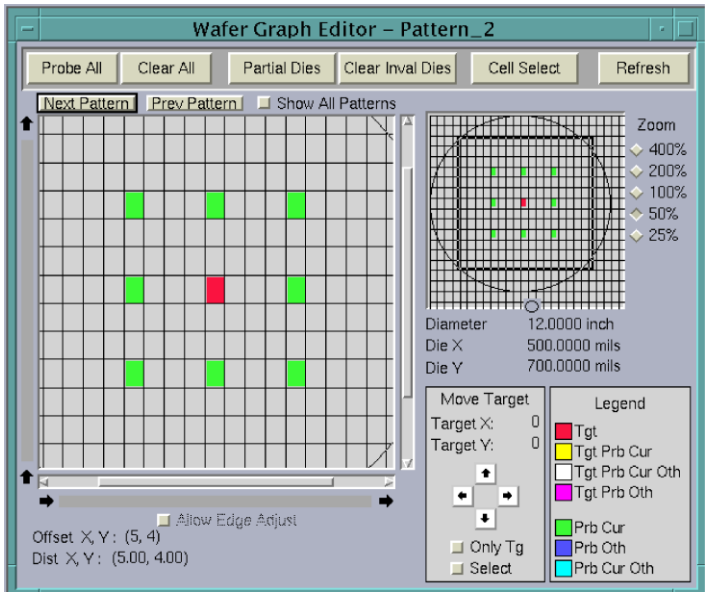
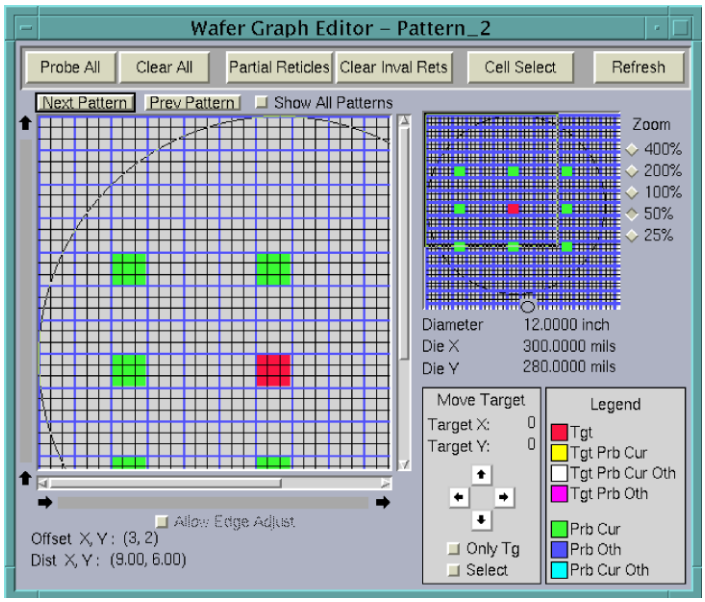


Figure 38: Wafer Graph Editor when Sites are Reticles selected in Wafer Setup Window



Note that in the previous figure, the graphical site display at left shows both the full reticles and the individual dies within each reticle.

Large control buttons

Probe All: Selects all possible sites on the wafer.

Clear All: Clears all selected sites on the wafer.

Complete Dies/Partial Dies: Visible only when Sites are Dies is selected in the Wafer Setup window; selects whether partial dies or only complete dies can be selected for testing.

- **Partial Dies:** You can select dies that do not fall completely within the usable wafer space.
- **Complete Dies:** You cannot select dies that do not fall completely within the usable wafer space (for example, a Probe All action will then select only dies that are fully within the usable wafer space). If any previously selected dies straddle usable wafer boundaries, those dies are deselected when you click Refresh or Clear Inval Dies.

Complete Rets/Partial Rets: Visible only when Sites are Reticles is selected in the Wafer Setup window; selects whether partial reticles or only complete reticles can be selected for testing.

- **Partial Reticles:** You can select reticles that do not fall completely within the usable wafer space.
- **Complete Reticles:** You cannot select reticles that do not fall completely within the usable wafer space (for example, a Probe All action selects only reticles that are fully within the usable wafer space). If any previously selected reticles straddle usable wafer boundaries, those reticles are deselected when you click Refresh or Clear Inval Rets.

Clear Inval Dies: Visible only when Sites are Dies is selected in the Wafer Setup window; when Complete Dies has been previously selected, clears all the dies selected for testing that do not fall completely within the usable wafer space.

Clear Inval Rets: Visible only when Sites are Reticles is selected in the Wafer Setup window; when Complete Rets has been previously selected, clears all the reticles selected for testing that do not fall completely within the usable wafer space.

Cell Select/Row Select/Col Select: Lets you select test sites by individual site, entire row, or entire column. Clicking the button scrolls through the selections.

Refresh: Refreshes the Wafer Graph Editor window with all changes or updates made to the data from all the other windows.

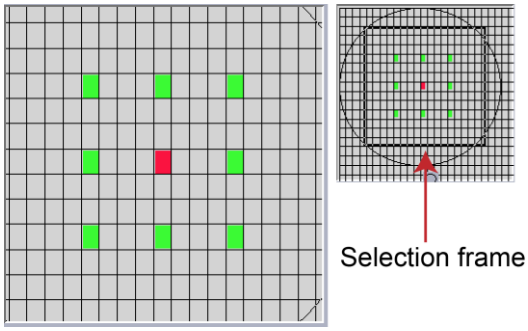
Graphical editing features

The wafer Graph Editor Window has the following graphical editing features.

Graphical site displays

In the following figure, the specific sites represented in the large pattern at left correspond to the sites inside the selection frame of the small pattern at right. The size of the selection frame is determined by the Zoom settings, as discussed in Zoom settings (on page 6-29). The selection rectangle can be moved with the mouse to any location on the wafer. Using a combination of Zoom settings and selection rectangle positions, the sites on any part of the wafer can be viewed and edited at different magnifications.

Figure 39: Graphical site displays

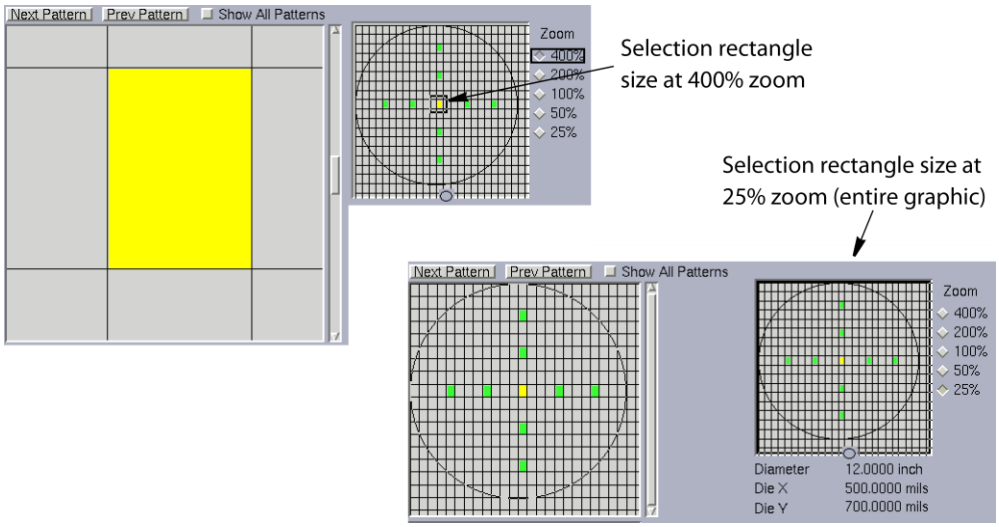


Target sites in the large pattern on the left can be directly selected or deselected by clicking. Target sites can be manipulated by the arrows and selections at the lower right of the Wafer Graph Editor. Refer to Move target arrow buttons and location display (on page 6-35).

Zoom settings

The Zoom settings at the far right control the size of the selection frame and the magnification of the sites displayed in the large graphical pattern. See the following figure.

Figure 40: Using Zooms



Controls located above the graphical site displays

- **Next Pattern:** Selects the next pattern in the Probe Pattern Editor's sequential list of patterns to be the current pattern.
- **Prev Pattern:** Selects the preceding pattern in the Probe Pattern Editor's sequential list of patterns to be the current pattern.
- **Show All Patterns:** Selects simultaneous display of all probe patterns in the `.wdf` file. The following figures illustrate the Wafer Graph Editor when Show All Patterns is selected.

Figure 41: Effects of selecting Show All Patterns for a Sites are Dies

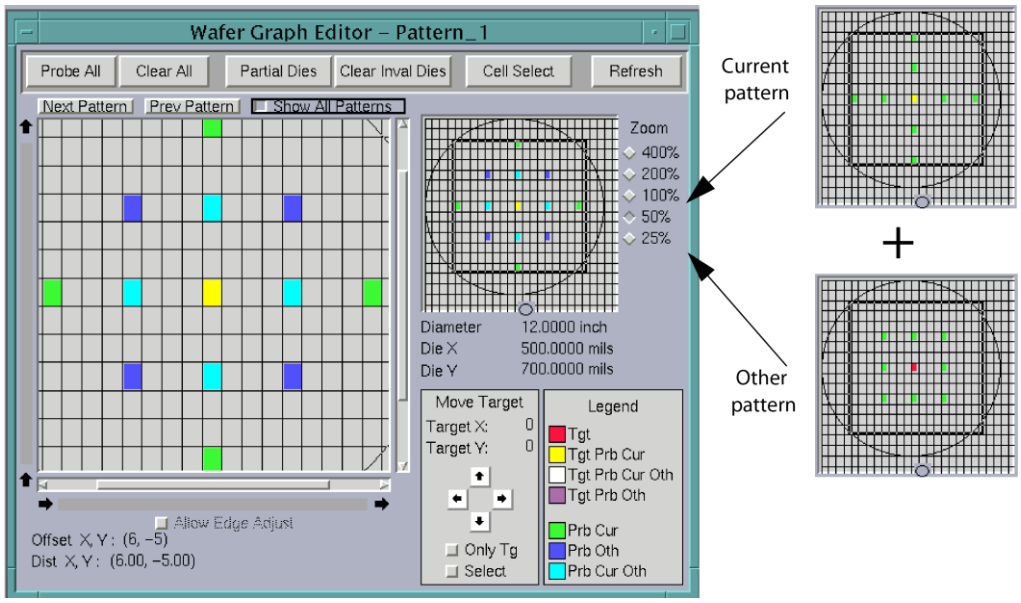
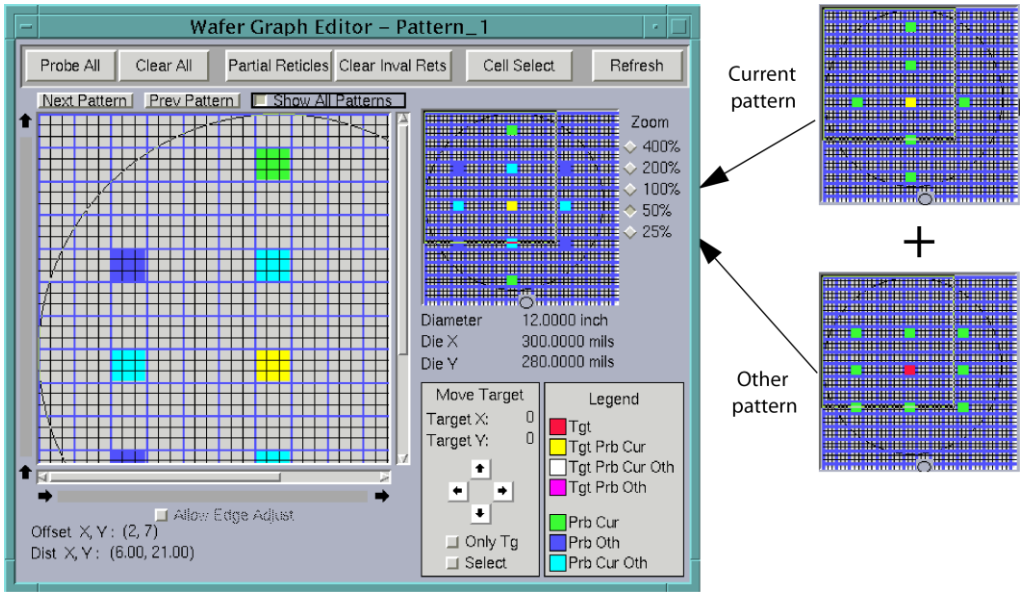


Figure 42: Effects of selecting Show All Patterns for a Sites are Reticles



Note that color codes distinguish the sites in the current probe pattern from the sites in other probe patterns. The color codes in previous figures show that the other (noncurrent) probe pattern in this example has a rectangular configuration, as defined by the blue-colored and cyan-colored sites. The cyan-colored sites are locations where current-pattern sites overlap other-pattern sites. The next subsection describes the color codes in detail.

Controls located below the graphical site displays

- **Allow edge adjust:** This allows the relative positioning of the die grid on the wafer to be shifted up to one die size in both the X and Y directions. Shifting the wafer outline does not affect any of the information used for the test process. It is intended for display purposes only and is not used by the system during wafer testing.
- **Offset X,Y and Dist X,Y:** These fields show the current position of the cursor arrow relative to the target site. Offset is in site (die or reticle) size units and Dist. is in mils or millimeters.

Legend

The site colors in the graphical site displays are coded as shown in the legend. This subsection explains the meaning of each color, as follows:

- **Tgt (red):** Target-only site (not tested).
- **Tgt Prb Cur (yellow):** Target site that is tested in the current probe pattern.
- **Tgt Prb Cur Oth (white):** Target site that is tested in the current probe pattern, overlapping at least one other target site that is tested in another probe pattern (applicable only when Show All Patterns is selected). See the following figures.

Figure 43: Legend

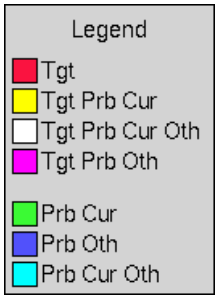


Figure 44: Scenario that results in Tgt Prb Cur Oth (white) color-coded target site

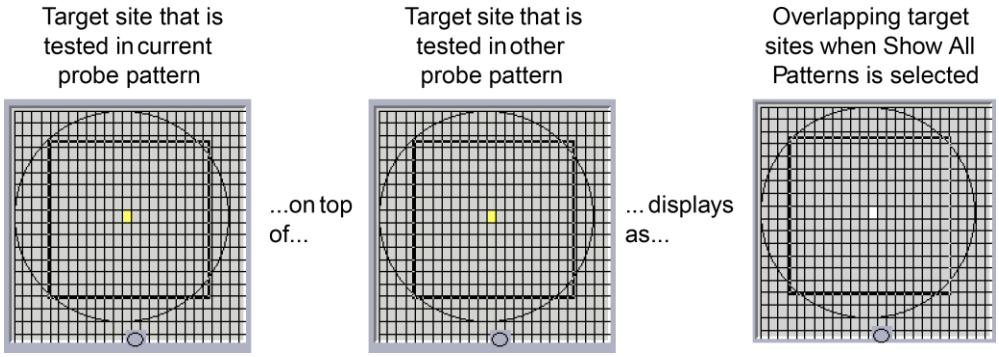
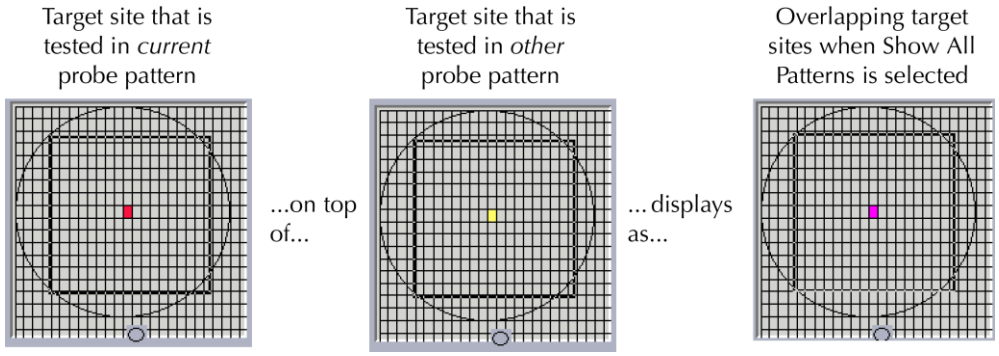


Figure 45: Scenario that results in Tgt Prb Oth (magenta) color-coded target site

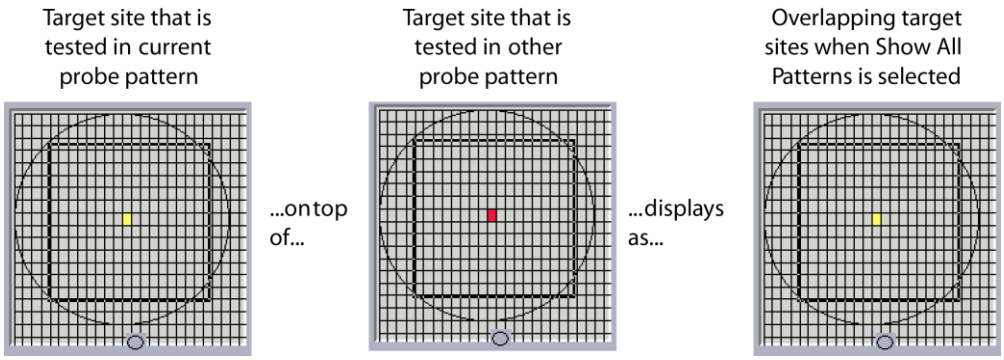


- **Tgt Prb Oth (magenta):** Target-only site (not tested) in the current probe pattern, overlapping at least one other target site that is tested in another probe pattern (applicable only when Show All Patterns is selected). See the following figure.

NOTE

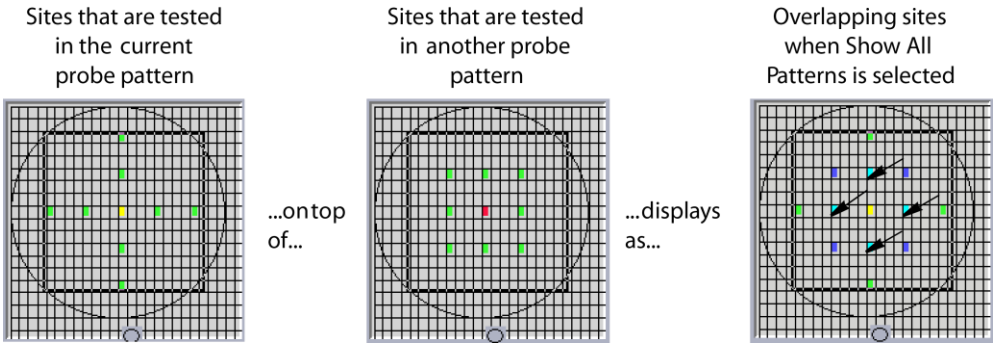
When a tested target site in the current probe pattern overlaps an untested target site or sites in another probe pattern, the target site color is yellow. See the following figure.

Figure 46: Scenario that results in Tgt Prb Oth magenta color-coded target site



- **Prb Cur (green):** Probed (nontarget) site that is tested in the current probe pattern.
- **Prb Oth (blue):** Probed (nontarget) site that is tested in another probe pattern. Applicable only when Show All Patterns is selected.
- **Prb Cur Oth (cyan):** Probed (nontarget) site that is tested in the current probe pattern, overlapping one or more probed sites that are tested in another probe pattern. Applicable only when Show All Patterns is selected. See the following figure.

Figure 47: Example of overlapping probed sites in current and non-current probe patterns



Move target arrow buttons and location display

The Target X and Target Y values display the target site X,Y location. The arrow buttons let you change this location in one of the following ways.

Move the target site by clicking the arrows:

1. Select the **Only Tgt** button, which allows only the target site to move.
2. Click an arrow button to move the target site one position in a given direction.

Move the target site by selecting a destination:

1. Select both the **Only Tgt** and **Select** buttons.
2. Select a new target site location by clicking it on the graphical display. Only the target site moves to the new location.

Move all selected sites by clicking the arrows:

1. Select the **Select** button.
2. Click the arrow buttons to move both the target site and the prober sites together to a new position.

Wafer information area

This area, located below the small graphical site display, contains the following information for the current `.wdf` file:

- Wafer diameter
- Die X and Y dimensions

Site name/subsite name data fields

Site names are not necessary for a single project wafer because all the sites on the wafer are the same.

A site name must be entered for a multiproject wafer before any subsites can be added. A site description can be added to each site name.

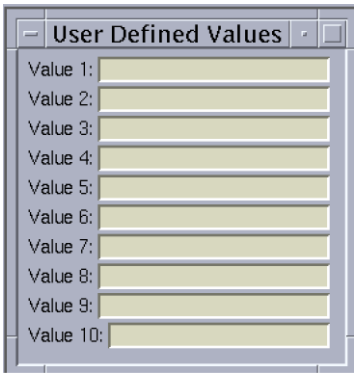
X and Y offset fields

These fields list the subsite X and Y offsets within each site. The values are in mils or millimeters, depending on the units selected in the Wafer Setup window.

User-Defined Values window

The User-Defined Values window contains the fields shown in the following figures.

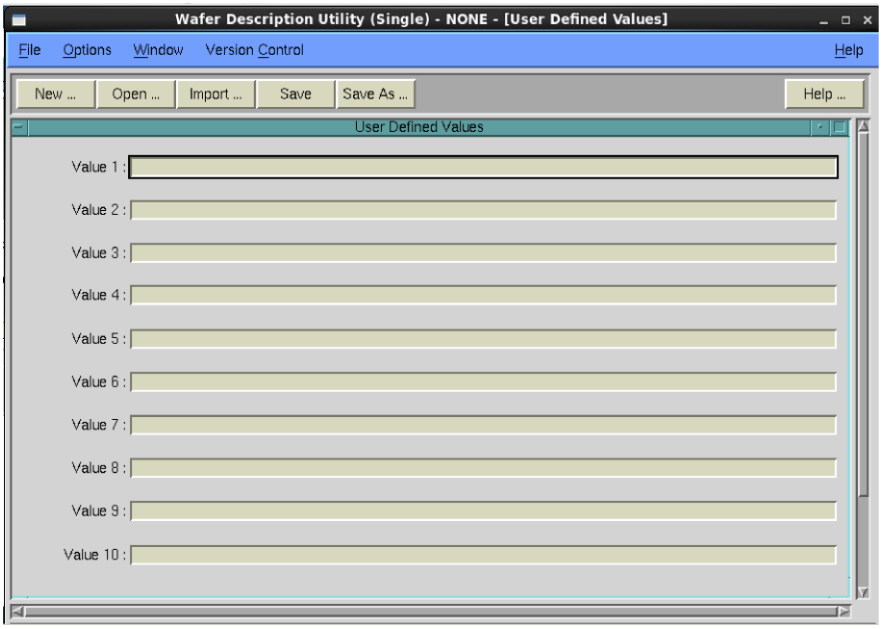
Figure 48: User Defined Values window, as it first appears



NOTE

If this window is not visible among the layers of Wafer Description Utility (WDU) windows, bring it forward by clicking the middle mouse button and selecting **User Defined Values** from the menu that appears.

Figure 49: User Defined Values window, expanded



In these fields, optionally enter information that you would like to be available to the Keithley Test Execution Engine (KTXE). For example, enter parameters for the user access point (UAP) code that you write. The information must meet the following requirements:

- **Data type:** Character (char) string without the quotation marks
- **Maximum string length:** 128 characters

The WDFRec pointer in the data pool contains these user-defined values. This pointer can be used in UAP routines after KTXE loads the wafer description file.

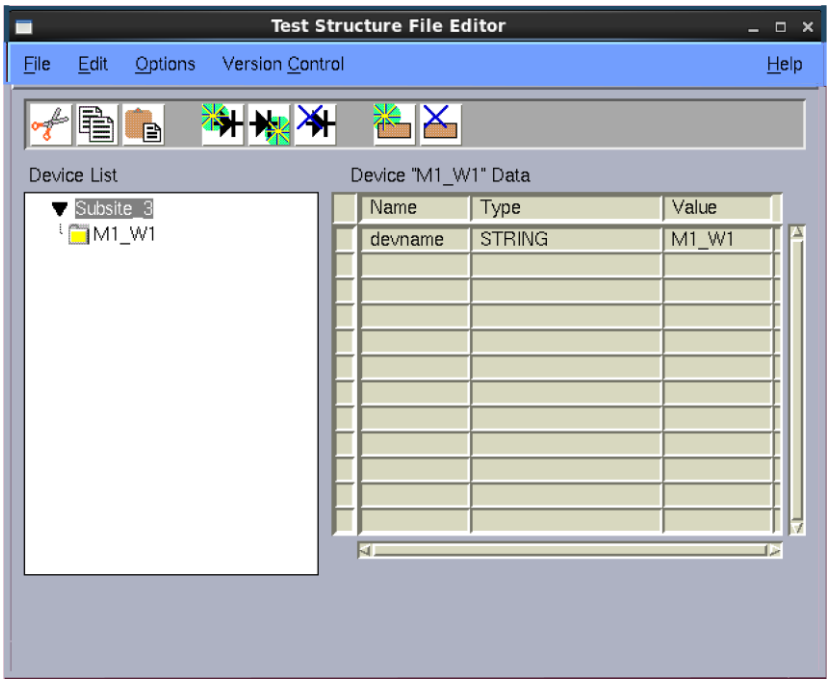
Test Structure File Editor (TSE)

The Test Structure File Editor (TSE) allows you to create parameters for a specific device located at a wafer subsite previously created using the Wafer Description Utility (WDU). The device parameters can then be plugged in with the test macro parameter data to create a more precise test that takes into account the specifications of the device.

TSE main window

The following figure shows the Test Structure File Editor (TSE) main window that appears on screen when the TSE is first activated. A brief description of each feature follows.

Figure 50: Test structure file editor main window



Title bar

The title bar lists the name of the currently open test structure file and the read-write status of the file.

TSE File menu

New: Clears the editor screen of all data.

Open: Opens a currently existing test structure file.

Launch Template File: Opens a new instance of the Test Structure File Editor (TSE) and loads the `template.tsf` file. This allows you to select devices from the template and copy or paste them into another instance of TSE to quickly add devices.

Save: Saves any changes made to the current test structure file.

Save As: Requests a filename before saving the current test structure file.

Exit: Exits TSE.

TSE Edit menu

Cut: Removes highlighted text, which can be restored to a new location using the Paste function.

Copy: Copies highlighted text, which can be placed in a new location using the Paste function.

Paste: Places cut or copied text in a new location.

Insert Device: Adds a device to the Device List.

Delete Device: Removes a device from the Device List.

Add Info After: Adds device information after the currently selected data.

Delete Info: Deletes the currently selected data.

Each of the items in the Edit menu also has a corresponding button on the toolbar, which is located directly below the menu bar.

TSE Options menu

ID/Comment String: Opens a text box that allows you to enter additional information about the device.

TSE Help menu

TSE Documentation: Displays current Test Structure File Editor (TSE) help documentation.

About: Displays the software revision level.

Device list

The Device List is located on the left side of the main window. It provides a list of all of the device parameters that are available for the current subsite. Clicking one of the devices listed opens that device's parameters into the Device Data window.

Device data window

The Device Data window is located on the left side of the main window. This window is a table that lists all of the parameters that apply to the device selected in the Device List. The parameters in this window can be modified by clicking any of the cells.

Copying device data

Device data can be cut, copied, and pasted within the same instance of the Test Structure File Editor (TSE), or between separate instances.

To cut or copy device data:

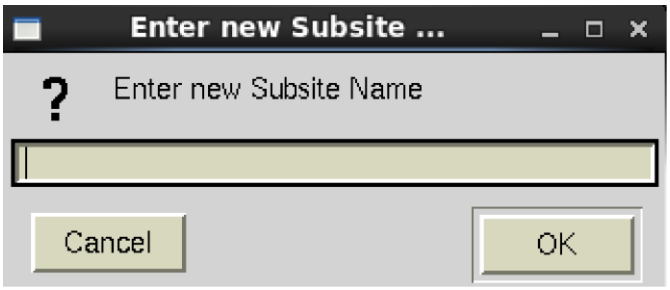
1. Select the device data and click the right mouse button. A Device Actions pop-up menu is displayed.
2. Select **Cut** or **Copy** to place the device data on the clipboard. **Cut** removes the device data from the current location and places it on the clipboard. **Copy** leaves the device data in its current location and only places a copy of it on the clipboard.
3. Select a new location in either the same instance of TSE, or in another instance, and right-click again. The Device Actions pop-up menu is displayed.
4. Select **Paste Before** or **Paste After** and the device data is inserted in the selected location.

Creating a test structure file

To create a test structure file:

1. Select **New** from the File menu. A text box, shown in the following figure, opens asking you for the Subsite name. Enter the name of a subsite that is currently in use in a wafer description file (`.wdf`) and click **OK**.

Figure 51: Subsite name test box



2. A device name text box opens. Enter the name of the device and click **OK**. To add additional devices to the current list, click the last device in the Device List and select the **Add Device After** icon or the menu item from the Edit menu. Repeat this process until you have added all of the devices required.

Figure 52: Device name text box

3. To enter device data, select the device from the Device List, then select the **Add Info After** icon or menu item. The first row already contains the default parameter name, `new_name`. Enter the parameter name to be used.
4. In the Type column, click the cell to activate the pop-up menu.
5. Click the arrow to the right of the cell to open the menu and select the proper parameter type. The parameter type, IDENTIFIER, references predefined probe-card or global-identifier data.

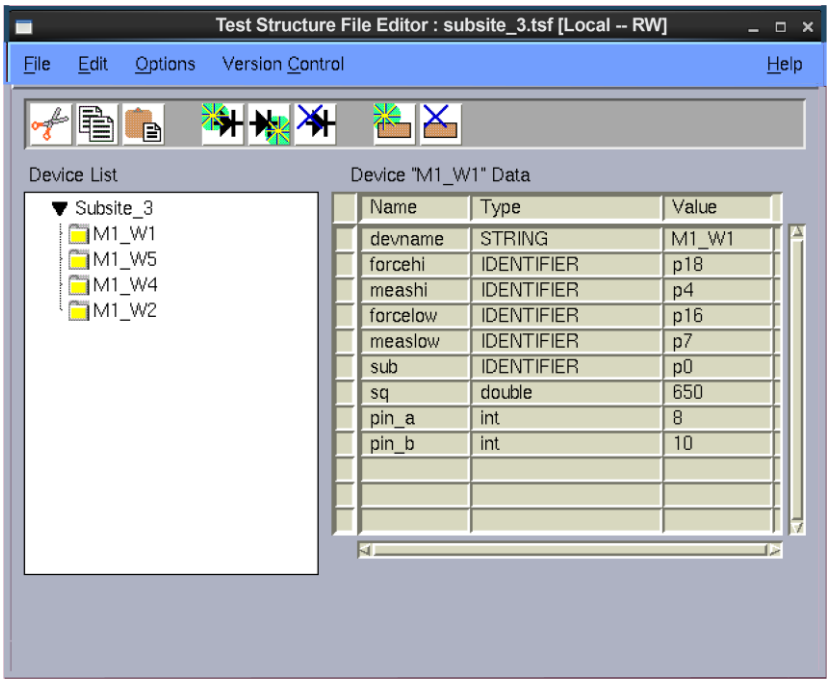
NOTE

IDENTIFIER values must start with a non-numeric character, and can only contain letters, numerals, and underscores.

6. In the Value column, enter the actual value for the parameter.
7. Once all of the device data for the subsite has been entered, select **Save** from the File menu.

   The test structure file (`.tsf`) is now complete, as shown in the following figure, and can be accessed by the Keithley Interactive Test Tool (KITT) Parameter Entry window if that subsite is selected in the KITT subsite box.

Figure 53: Completed test structure file



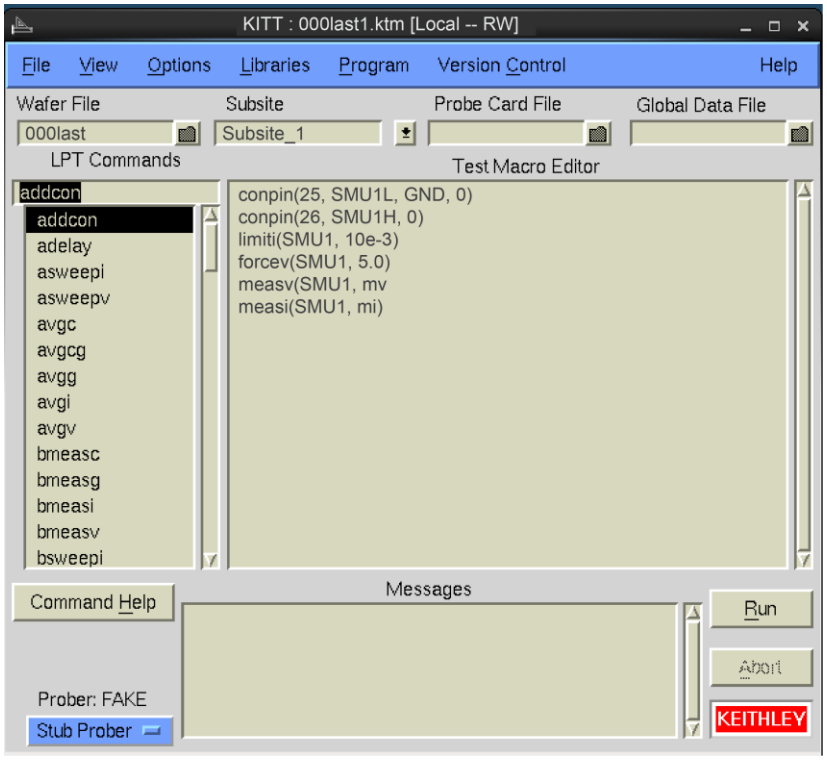
Keithley Interactive Test Tool (KITT)

The Keithley Interactive Test Tool (KITT) provides a software environment that gives you immediate access to standard test libraries. KITT provides language-independent calls to create test macros and allows interactive, immediate execution of test macros and prober calls. KITT is useful to diagnose instrument, prober, and device measurement problems and create test macros.

KITT main window

The following figure shows the Keithley Interactive Test Tool (KITT) main window that appears on the screen when KITT is first activated. A brief description of each feature follows.

Figure 54: KITT main window



Title bar

The title bar lists the name of the currently open test macro file and the read-write status of the file.

KITT File menu

The File menu items let you:

- Start a new Keithley Test Module (KTM).
- Open test macro files into the Test Macro Editor window.
- Include an existing test macro.

- Save test macro files shown in the Test Macro Editor window.
- Delete macro files.
- Print test macro files.
- Exit Keithley Interactive Test Tool (KITT).

KITT View menu

The View menu contains:

- **Command Help:** Opens or closes the Command Help window. This can also be done from the Help button in the parameter entry window or Keithley Interactive Test Tool (KITT) main window.
- **Parameter Entry:** Opens or closes the KITT Parameter Entry window for the presently selected test command.
- **Results:** Opens or closes the KITT Results window.
- **Keithley Data Editor:** Opens the Keithley Data Editor.
- **Results Settings:** Opens the Results Settings window where the plot, log, and user parameters can be set for the present macro.
- **Test Macro Description:** Lets you attach notes to the present test macro.

KITT Options menu

The Options menu controls:

- Whether the tester hardware is online or offline.
- If autoplotting is on and if Keithley Curve Analysis Tool (KCAT) will be used to plot the results, with the plot value set to X or Y.
- Whether or not results are grouped by result name in the Keithley Interactive Test Tool (KITT) results window.
- Determines whether just the filename or the entire path will be saved in the `.ktm` file. If the entire path is to be saved, it is recommended that the environment variables be used. This will make subsequent file retrieval easier.
- Whether the wafer and subsite data is saved with the `.ktm` file.
- Whether the probe card file is saved with the `.ktm` file.
- Whether the global data file is saved with the `.ktm` file.
- Whether or not to create a `kitt.ini` file to save the current status of the Options menu.
- Whether the execution time for the test macro and of test modules will be shown in the Messages window.
- Whether or not Probe Card values and Global Data values are resolved in the Parameter entry window.

- Whether or not Strict Parameter Resolution is enforced.
- Prober error and transaction logging, and viewing the error or transaction logs.
- The user-defined level of the error log.
- Whether or not the results are overwritten in the KITT results window when a test macro is run.
- Whether all of the results are shown in the KITT results window, or only those selected to be viewed in the Results Settings window.

KITT Libraries menu

The Libraries menu lets you select the following command libraries:

- Prober
- LPTLib
- User Libraries

KITT Program menu

The Program menu items let you:

- Run or loop program test macros.
- Generate C programming language code.
- Debug C programming language code before running the macro (practice task).
- Clear messages previously written in the Messages window.
- Remove error identifier strings from text in the Test Macro Editor window.

KITT Help menu

Help menu items display Keithley Interactive Test Tool (KITT) help documentation and software revision level.

Test data fields

These fields include:

- **Wafer File:** The wafer description file to be used during testing.
- **Subsite:** The wafer subsite the test macro is to be run on.
- **Probe Card File:** The probe card file used when the current test macro is run.
- **Global Data File**: The global data file used when the current test macro is run.

Right-clicking any of the data fields displays a pop-up menu. The menu allows you to open the data file in the appropriate tool for editing. For example, right-clicking the Wafer File field will allow you to open the wafer file in the Wafer Definition Utility (WDU).

This information is written into the macro file (`.ktm`) so future work in the Keithley Interactive Test Tool (KITT) can be continued without reentering these selections. However, the wafer files, probe card files, and global data files used in production runs by the execution engine are specified in the wafer plan and cassette plan editors. The execution engine uses the subsite information stored in the test macro (`.ktm`) to determine what subsite the macro is to run on.

Commands scroll box

The commands scroll box displays and allows selection of the various commands for the selected library. To display information about the selected command, click the **Command Help** button. A pop-up window will display pertinent information about the selected command including parameters that it uses.

Test macro editor window

The Test Macro Editor window is the primary work area for generating your test macros. Commands can be moved from the Parameter Entry window to the Test Macro Editor window using the OK and Apply buttons. Control buttons associated with the Test Macro Editor window are:

- **Run:** Runs the entire test macro listed in the Test Macro Editor window, or just the highlighted lines.
- **Abort:** Aborts the test macro presently being executed.

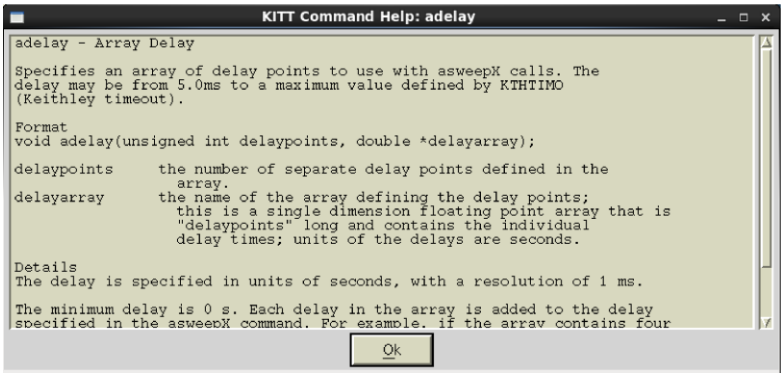
Messages window

The Messages window displays event and error messages that occur during Keithley Interactive Test Tool (KITT) operation. You can clear the Messages window by selecting the Clear Messages menu item in the Program menu.

Command help

The Command Help button will display help information for the test command currently selected in the Command Help window, as shown in the following figure.

Figure 55: Command help window



Prober

The Prober displays the name of the currently configured prober driver. You can toggle between Fake and Real Prober by clicking the Prober selection box. The commands supported by the currently configured prober driver will be displayed in the Commands box when Prober Library is selected under the Libraries menu.

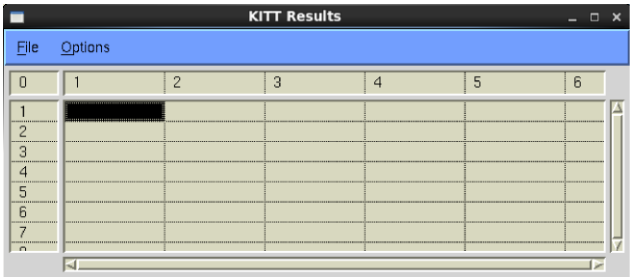
KITT Results window

The following figure shows the Keithley Interactive Test Tool (KITT) Results window. Results data from the test macros are displayed in a spreadsheet format. Each column displays one result when the test macro is run. The column heading identifies the result name. Each row is a separate execution of the macro.

KITT Results window menu items include:

- **File:** Lets you open, save, and print results data and clear the results data window.
- **Options:** Lets you select the delimiter to be used to separate individual data elements in the output file. This ASCII-delimited format allows data to be read into most spreadsheet and data analysis tools.

Figure 56: KITT results window



KITT math, array and logic expression support

Keithley Interactive Test Tool (KITT) supports the following functions.

Math

Refer to your C language reference manual for more information on proper syntax use.

Definition

Keithley Interactive Test Tool (KITT) supports the following math operators and functions:

- Addition '+'
- Subtraction '-'
- Multiplication '*'
- Division '/'
- Parentheses '(' and ')'

  The parentheses '(' and ')' operators may be used to define precedence of the math operations

- int abs(int val);

  abs() returns the absolute value of its int operand

- double fabs(double x);

  fabs(x) returns the absolute value of x

- double exp(double x);

  exp(x) computes the exponential function e**x

- double log(double x);

  log(x) computes the natural logarithm of x

- double log10(double x);

  log10(x) computes the base-10 logarithm of x.

- double pow(double x, double y);

  pow(x, y) computes x raised to the power y

The operators and functions may be used as part of a calculation to define a parameter value to a test. The operators and functions may act on data associated with identifiers from the following data sources:

- Predefined identifiers within a test macro file (`.ktm`)

- Global data from a global data file (`.gdf`)
- Probe card data from a probe card file (`.pcf`)
- Device data from a test structure file (`.tsf`)
- Literal values (for example, 1.23e-4)

The operators and functions above can also be used with the equal sign (=) assignment operator. This operator can assign a new value to previously defined identifiers or create a results identifier. If this result value is not previously defined, the result will be type DOUBLE.

Parameter values are resolved as each macro line is executed.

Restrictions

- Operators are not valid with parameter set identifiers.
- Parameters with operators will not be saved in parameter sets.
- Variable must not be named with a digit and an `E` or `e` as the last two characters.

  For example:

  The variable name `foo2e` is invalid.

  The variable name `Foo2` is valid.

  This restriction is due to the expression `Foo2e - 5` being confused with the number `2e-5`.

- A runtime error occurs if a divide by `0` is attempted. The macro execution only terminates at that point.

Examples

- Math in parameters

```
Test(param1, (R1/(ss_dev1_width * ss_dev1_length), output_result)
```

- Assignment operator

```
Area = ss_dev1_width * ss_dev1_length
AnotherResult = ((result_array[0] * 3.14)/some_identifier)
```

The results `Area` and `AnotherResult` will be type DOUBLE if not previously defined.

- The compiler requires the use of decimal points to signify floating-point notation.

  For example:

```
1/8 = 0
1.0/8.0 = 0.125
```

Array support

Arrays must be defined before their use. Definition can occur as:

- Predefined identifiers within a test macro file (`.ktm`)
- Global data from a global data file (`.gdf`)

A complete array or a single element can be an input or an output parameter. If a complete array is passed into or out of a test, the array index is not specified. If a single array element is passed into or out of a test, the array index is defined by using brackets `[` and `]`. An expression can be used to define the array index.

To use a variable as an array index, the variable must be type INT. This is necessary for the Practice Task and Save As C features to function correctly.

Restrictions

There are some minor restrictions on result array assignment notation. No nested array notation and no complex math expressions are allowed as index expressions. The following list of examples should help explain.

Supported:

```
gdfArray[1] = function()
pdiArray[x + 2] = function()
resultArray[3] = 4.53
```

Not supported:

```
AnyArray[(x + 2 ) * 3] = function()
AnyArray[anotherArray[3] + 1] = function()
```

Examples

Examples of how arrays can be used in the Keithley Interactive Test Tool (KITT) are described in the following topics.

Array elements as outputs

The array `result_array` is defined previously as array data. The `result_array[]` parameters are output parameters. The complete `result_array` parameter is passed into the `Fit_function`. The `number_of_elements_in_array` parameter indicates to the function how many array elements to use.

```
Test(ss_dev1_pad1, ss_dev1_pad2, input_item, result_array[0])
Test(ss_dev2_pad1, ss_dev2_pad2, input_item, result_array[1])
Test(ss_dev3_pad1, ss_dev3_pad2, input_item, result_array[2])
Test(ss_dev4_pad1, ss_dev4_pad2, input_item, result_array[3])
Fit_function(result_array, number_of_elements_in_array, result_value)
```

Also, arrays may contain function return values.

```
Result_array[1] = Test(param1)
```

Array elements as inputs

The array `input_array` is defined previously as array data or as a result output from a function. In the following example `resulta`, `resultb`, and `resultc` are output parameters.

```
TestB(ss_dev1_pad1, ss_dev1_pad2, input_array[0], resulta)
TestB(ss_dev1_pad1, ss_dev1_pad2, input_array[1], resultb)
TestB(ss_dev1_pad1, ss_dev1_pad2, input_array[2], resultc)
```

Also,

```
Test(param1, input_array[(identifier*2)], result_array[identifier])
```

Array as Output

In the following example, `result_array` is an output parameter and `number_of_elements_in_array` is input parameter.

```
TestSweep(param1, param2, result_array, number_of_elements_in_array)
```

This is already supported in version 3.2 and later.

Array as Input

In the following example, `result_array` and `number_of_elements_in_array` are input parameters.

```
Fit_function(result_array, number_of_elements_in_array, result_value)
```

This is already supported in version 3.2 and later.

If-then-else and logical expressions

The Keithley Interactive Test Tool (KITT) supports the if-then-else statement with the following Boolean operators for int, float, and double data types:

- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to
- == equal to
- != not equal to

The following Boolean operators are supported for int data types:

- && logical AND
- || logical OR
- ! logical NOT

Each of the Boolean operators generates a 0 or 1 response. If a result is created and assigned using a Boolean operator, the result type is treated as a double if the result has not been previously defined otherwise.

Restrictions

- When using the `if` statement, the open and close braces must be used and they must be on separate lines.

Example:

```
if (a > b)
{
	value = 4
}
else
{
	value = 0
}
```

- Nesting is supported up to 32 levels.

Using generated identifiers in a test macro

Generated identifiers are the following types:

- Test Structure Data Identifier
- Parameter Set Data Identifier
- Generated Result Identifier

Test Structure Data Identifiers

Test Structure Data Identifiers take the form `device_parameter`.

The device part of the identifier is set in the Keithley Interactive Test Tool (KITT) Parameter Entry window by selecting the device to use at a subsite.

The parameter part of the identifier is associated with the name of the parameter.

These Test Structure Data Identifiers are only generated for parameters that have the same name as an item in the selected device's test structure file. The test structure file used is based on the subsite name set in the KITT main window.

Parameter Set Data Identifiers

Parameter Set Data Identifiers take the form `module_paramset_parameter`.

The `module` part of the identifier is the name of the test module.

The `paramset` part of the identifier is the name given to the parameter set.

The `parameter` part of the identifier is associated with the name of the parameter.

Selecting a parameter set in the Keithley Interactive Test Tool (KITT) Parameter Entry window places the parameter set data identifiers in all symbolic value fields that have a matching parameter. If a parameter's symbolic value field contains test structure data and a parameter set contains a matching parameter, the parameter set data identifier will replace the test structure data identifier in the field.

Generated Result Identifiers

Generated Result Identifiers take the form `subsite_device_parameter`.

The `subsite` part of the identifier is the subsite name set in the Keithley Interactive Test Tool (KITT) main window.

The `device` part of the identifier is the device selected in the KITT Parameter Entry Window.

The `parameter` part of the identifier is the output parameter name.

Custom Identifier Separators

The generated identifier separator may be changed from the default _ to any single letter `a` through `z` or `A` through `Z`. This is set in the `kitt.ini` file.

KTE data usage

The following topics describe Keithley Test Environment (KTE) data usage.

Data pool

- A data storage area generated during Keithley Test Execution Engine (KTXE) and Keithley Interactive Test Tool (KITT) execution.
- Run-time storage of probe card file (`.pcf`) data, global data file (`.gdf`) data, global predefined date identifier (PDI) data, KTXE control data, and data generated with `dpAdd` functions. Each run starts with a clean data pool.

Data precedence

The Keithley Test Execution Engine (KTXE) and Keithley Interactive Test Tool (KITT) resolve parameters from the data sources in the following order:

- Local predefined data identifiers (PDI)
- Test structure data
- Parameter set data
- Probe card file data
- Global data file data
- Global PDI
- Generated results

Data sources

The following data sources are used in the Keithley Test Environment (KTE).

Local predefined identifiers

- A local predefined identifier (PDI) is a value local to a Keithley test module (KTM).
- A local PDI is highest in the data precedence. A local PDI value is used before the data pool.
- Useful for local constants and defining data types of intermediate variables.

Test structure data

- Loaded into internal test structure memory.
- Used to define test structure and device-specific data.
- Changing test structure data effects all test modules that use the data as a parameter.
- Use to define pad-to-structure mapping. Example: Drain = pad3.
- Use to define test structure characteristics. Example: Length = 100.

Parameter set data

- Loaded into internal parameter set memory.
- Used as templates for standard test parameters.
- Changing a parameter set permits modification of all Keithley test modules (KTMs) that use the parameter set.

Probe card file data

- Probe card file (PCF) data is loaded into the data pool during the wafer-loading phase of the Keithley Test Execution Engine (KTXE).
- Because the PDF file data is loaded for each new wafer plan, items from previous PCF files may be in the data pool.
- Keithley Interactive Test Tool (KITT) loads the PCF file at the start of each run.
- Use to define tester pin-to-pad mapping. Example: Test pin 8 = pad3.

Global data file data

- Global data file (GDF) data is loaded into the data pool during the cassette plan-loading phase of the Keithley Test Execution Engine (KTXE).
- The GDF files are loaded in the order of definition. If two GDF files load items of the same name, the value in the last file is used.
- The Keithley Interactive Test Tool (KITT) loads the GDF file at the start of each run.
- Useful for passing data between Keithley test modules (KTMs).
- Useful for passing global test parameters into tests.
- KTXE supports loading of multiple global data files.
- Can be used to change execution behavior of KTXE.
- User access points (UAPs) can access this data.

Global predefined identifiers

- A global predefined data identifier (PDI) value is available to the defining Keithley test module (KTM). After KTM execution, they are placed into the data pool. Any KTM executed following the defining KTM or re-execution of the defining KTM gets the value of the global PDI from the data pool.
- A global PDI is lowest in the data precedence.
- Useful for passing data between KTMs (global data is more flexible).

Generated results

- Output variables generated by the Keithley Test Execution Engine (KTXE) or the Keithley Interactive Test Tool (KITT).

Keithley Test Execution Engine control data

- The Keithley Test Execution Engine (KTXE) places items in the data pool at run time. This data is used internally by KTXE and may be altered to change KTXE execution behavior.
- See Data pool (on page 6-228) for a list of these items.
- During Keithley Interactive Test Tool (KITT) execution, no KTXE control data exists.

Passing data between Keithley test modules

Data is passed between Keithley test modules (KTMs) through the data pool. KTMs may place data into the data pool by outputting the results to a global data file (GDF) defined name or a global predefined identifier (PDI), or using a `dpAdd` function in a user library. Data pool items accessed by a KTM must exist before executing a macro line. If not, a run-time error occurs and the KTM currently executing terminates.

Example:

Two KTMs each generate one result that is used by a third KTM. Two values are placed in the global data file that is loaded in the data pool.

Global data file: `passedresults.gdf`

```
slope_5x20, DOUBLE_P, 0
slope_5x10, DOUBLE_P, 0
```

KTM: `x20.ktm`

```
nvth5x20 = vth_ext(p10,p11,p13,p24,0.1,0.0,2.0,1e-6,100, slope_5x20)
```

KTM: `x10.ktm`

```
nvth5x10 = vth_ext(p9,p11,p13,p24,0.1,0.0,2.0,1e-6,100, lope_5x10)
```

KTM: `calc.ktm`

```
deltaw(slope_5x20, 20, slope_5x10, 10, result)
```

Use of ibupu functions in the Keithley Interactive Test Tool

In the Keithley User Library Tool (KULT) user library routines, the `ibupu()` functions are called using the following syntax:

```
ibupu(unit, IBUPU_CLEAR, slot)
ibupu(unit, IBUPU_DEFINE, slot, tad, lad, sad, rmd, eod, wmd)
ibupu(unit, IBUPU_FINISH)
ibupu(unit, IBUPU_LOCAL, slot)
ibupu(unit, IBUPU_READ, slot, *read_buff, count)
ibupu(unit, IBUPU_SRPOLL, slot)
ibupu(unit, IBUPU_SRQWAIT, slot, timeout)
ibupu(unit, IBUPU_TRIGGER, slot)
ibupu(unit, IBUPU_WRITE, slot, *write_buff, length)
```

When using these functions from the Keithley Interactive Test Tool (KITT), use the following syntax:

```
ibupu_clear(unit, slot)
ibupu_define(unit, slot, tad, lad, sad, rmd, eod, wmd)
ibupu_finish(unit)
ibupu_local(unit, slot)
ibupu_read(unit, slot, *read_buff, count)
ibupu_remote(unit, slot)
ibupu_srpoll(unit, slot)
ibupu_srqwait(unit, slot, timeout)
ibupu_trigger(unit, slot)
ibupu_write(unit, slot, *write_buff, length)
```

Parameter Set Editor (PSE)

The Parameter Set Editor (PSE) allows you to add, remove, or edit parameter set data assigned to a library module. The PSE does not modify the default data created in the Keithley User Library Tool (KULT), but allows you to create additional parameter sets that can be used when creating a test macro.

New parameter sets can be created by pressing the **Save** button on the Keithley Interactive Test Tool (KITT) parameter entry screen. If the Parameter Set Editor is active when this save occurs, the Parameter Set Editor is not refreshed with the new data until the specified module name is reopened. Clicking the folder icon for the module refreshes the display.

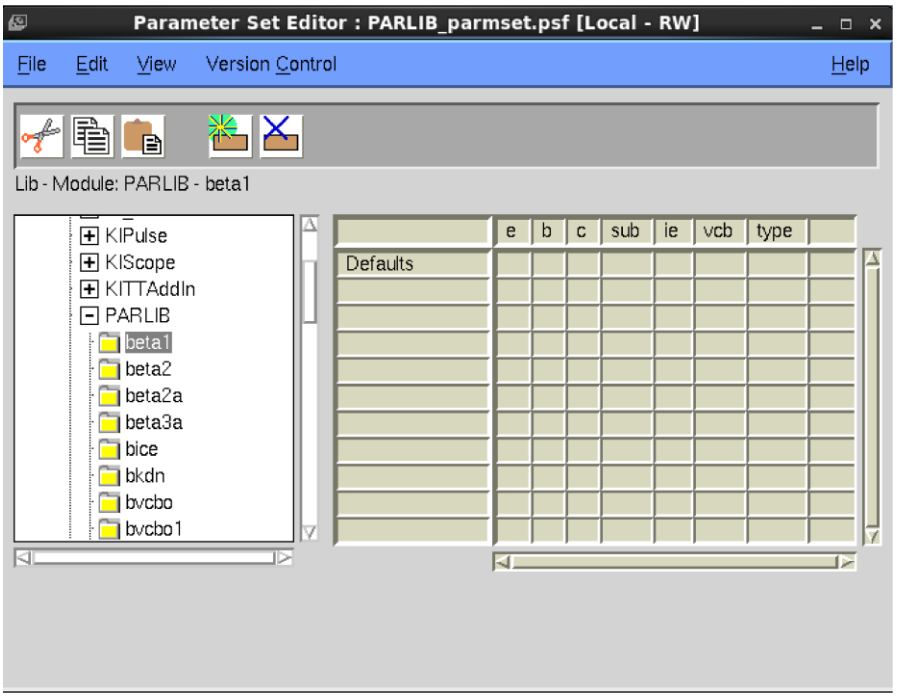
PSE main window

There are two main areas of the Parameter Set Editor (PSE), shown in the following figure.

The Library Browser: Allows you to navigate through a directory tree of available user libraries and modules to select the parameter data you want.

The Parameter Data list box: Lists all of the available parameter data for a module that is selected from the library browser.

Figure 57: PSE main window



Title bar

The title bar lists the name of the currently open parameter set file and the read-write status of the file.

PSE File menu

The Parameter Set Editor (PSE) File menu contains the following items:

- **Save**: Saves any changes made to the parameter set data.
- **Exit**: Closes the PSE.

PSE Edit menu

The Parameter Set Editor (PSE) Edit menu contains the following items:

- **Cut**: Removes highlighted text, which can be restored to a new location using the Paste function.
- **Copy**: Copies highlighted text, which can be placed in a new location using the Paste function.
- **Paste**: Places cut or copied text in a new location.
- **Add New Set After**: Adds a new set of parameters after the currently selected set.
- **Delete Parameter Set:** Deletes the current parameter set.

Each of the items in the Edit menu also has a corresponding button on the toolbar, which is located directly below the menu bar.

PSE View menu

- **Parameter Sets**: Toggles the Parameter Set Editor between showing the entire main window and just the Library Browser.

PSE Help menu

- **PSE Documentation**: Displays current Parameter Set Editor (PSE) help documentation.
- **About**: Displays the software revision level.

Library browser

To navigate through the available libraries, click the plus (**+**) icons to expand the library tree and the minus (**-**) icons to collapse the library tree. A folder icon indicates a user library module that could contain parameter set data. If parameter set data exists for a module, clicking the folder icon opens the parameter data contained in the selected module in the Parameter Data list box.

If the directory tree is longer than the viewable window, a vertical scroll bar appears that can be used to scroll the library or module list into view. Holding the left mouse button down and dragging on an empty section of the directory tree background also allows you to move the view area.

Parameter data list box

NOTE

The default data for a parameter cannot be modified from the Parameter Set Editor; it can only be changed using Keithley User Library Tool (KULT).

The first row of the parameter set edit panel lists the column heading for the library module: Parameter set name and parameter names.

The first column displays the parameter set name. If the library has defaults set for any parameters, the parameter set name `Defaults` will displayed in the first column with the values in the appropriate field. Parameter set names may contain any alphanumeric character or an underscore character. The name of the parameter set is made available in the Parameter Set Selection box in the Keithley Interactive Test Tool (KITT) Parameter Entry window.

The second and greater columns display the parameter values or identifiers. Valid values must be appropriate for the data type of the parameter. Identifiers must be resolvable by KITT and the Keithley Test Execution Engine (KTXE).

This field displays a message about the item that has focus. For parameter set parameters, the following items are presented: Parameter data type and value range.

To edit data in one of the cells, click the cell and enter the new value. To remove a parameter set data value, clear the contents of the cell, or select **Cut** from the Edit menu.

To add a new parameter set, click the row above where you wish the new parameter set to be. Select **Add New Set After** from the Edit menu. A new parameter set is created. Enter the name of the new parameter set in the first column. Enter the rest of the parameter set data as needed.

To delete an entire parameter set, click the parameter set name cell and select the **Delete Parameter Set** button. You are asked to confirm the deletion. Please note that the Cut option does not function for entire parameter sets.

To refresh the library listing to view new or remove deleted libraries and modules, minimize and then maximize the root directory labeled Libraries in the library browser area.

If a module is deleted from a library, all parameter set data is purged from the data structures. You must save this parameter set to completely remove the data from future use.

Keithley User Library Tool (KULT)

The Keithley User Library Tool (KULT) is a tool used to create and manage user modules and libraries. It performs a different function than the other Keithley Test Environment (KTE) tools. The other tools are used to capture data to describe the wafer, tests, parameter limits, wafer test plans, cassette test plans, and operator test selections. KULT is used to make code additions and organize the code into modules and libraries. In KULT, you can edit and compile C language code.

KULT is used to extend the testing capabilities of a standard software distribution. A structured graphical user interface is provided to facilitate the code and data entry. The module calling structure, code, parameters, description, and include files are all entered separately. This structured entry allows the user extensions to be functionally integrated with the KTE test plan development and production testing tools. This means that proprietary test algorithms, shop floor control procedures, and other unique site requirements can be added without vendor-supplied software upgrades. Also, KULT-created user library extensions are compatible with future software upgrades.

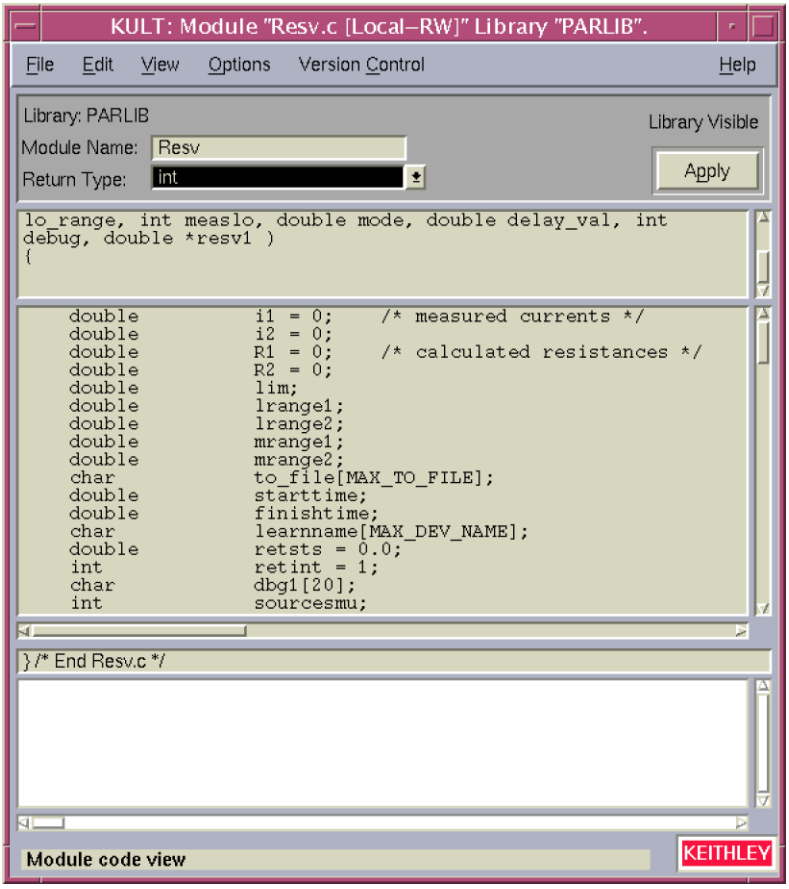
KULT libraries are created to allow for easy and quick extensions to the capabilities of the parameter test routines (PAR), prober drivers, and the production test execution engine (KTXE). Once a user library is developed, it is available for test macro development in the Keithley Interactive Test Tool (KITT). Or, the module can be included in KTXE for execution at the user access point (UAP) specified in the cassette plan.

KULT libraries are dynamically loaded when used by KITT and KTXE. All the file control benefits of libraries are realized (single master copy, reuse, and control). Site-specific test modules can be created once and then called by many different product test plans. A new user library can call the system libraries and other user libraries.

KULT main window

The Keithley User Library Tool (KULT) main window is shown in the following figure. The library name, module name, module call, and code are presented to the user in this window. Other windows are used to present the module's arguments and description.

Figure 58: KULT main window



Title bar

The title bar lists the name of the currently open library module and the read-write status of the file.

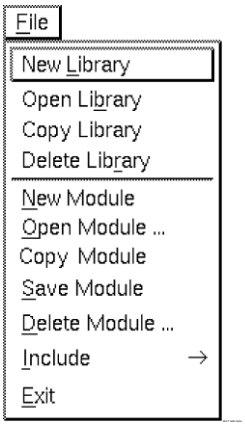
KULT File menu

The File menu shown in the following figure contains the following items:

- **New Library**: Produces the new library name dialog box and initializes the new library.
- **Open Library**: Lists available user libraries.
- **Copy Library**: Permits creating a copy of an existing library.
- **Delete Library**: Deletes the selected library and all of its contents.
- **New Module**: Produces the module name dialog box and initializes the new module.

- **Open Module**: Lists available modules.
- **Copy Module**: Permits copying the current module into an existing library.
- **Save Module**: Saves changes made to a module.
- **Delete Module**: Deletes the selected module from a library. Note that you must rebuild the library to ensure the module has been deleted.
- **Include**: Inserts other modules into the module body area. This function lets you alter an existing module or create a new module.
- **Exit**: Exits the Keithley User Library Tool (KULT).

Figure 59: KULT file menu



The Edit menu shown in the following figure contains file-editing functions.

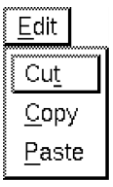
The edit options are:

- **Cut**: Removes highlighted text, which can be restored to a new location using the Paste function.
- **Copy**: Copies highlighted text, which can be placed in a new location using the Paste function.
- **Paste**: Places cut or copied text in a new location.

NOTE

When cutting, copying, and pasting between the main window and the Description window, use the Cut or Copy commands from the window that contains the text, and use the Paste command from the window where the text is being placed.

Figure 60: KULT edit menu



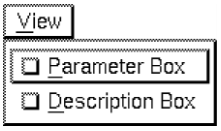
The Main Body Area, Description Window, and Include Files window all support the following menu items from the menu that appears when the middle mouse button is pressed:

- **Cut**: Same as Cut in the Edit menu.
- **Copy**: Same as Copy in the Edit menu.
- **Paste**: Same as Paste in the Edit menu.
- **Select All**: Highlights all text in selected window for cutting or copying.
- **Search**: Opens a dialog box that lets you specify a text string to search for.
- **User Paste Strings**: Reveals Add User Paste Strings, which lets you enter a text string that you want to paste into the selected window.

KULT View menu

The View menu lets you display or conceal the parameter box or the description box. See the following figure for the View menu options.

Figure 61: KULT view menu

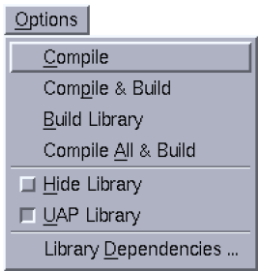


KULT Options menu

The Options menu shown in the following figure contains the following functions:

- **Compile:** Compiles source files into object files and checks for errors in the module.
- **Compile and Build**: Performs both the Compile and Build Library functions.
- **Build Library:** Creates a user library and establishes help files for the module.
- **Compile All and Build:** Performs the Compile and Build Library operations for all modules that are part of the current library.
- **Hide Library:** Toggles the availability of the presently open library to a test macro, in the Keithley Interactive Test Tool (KITT), between visible and hidden.
- **UAP Library:** Toggles the availability of the presently open library for use at a user access point (UAP).
- **Library Dependencies**: Allows you to specify other user libraries to link to.

Figure 62: KULT options menu



KULT Help menu

The Help menu contains online help information about the Keithley User Library Tool (KULT).

Module identification area

The module identification area is located directly below the menu bar and defines the active library and module. The components of this area are:

- **Library Name:** Defines the active library.
- **Module Name:** Defines the active module.
- **Return Type:** Defines the format of the module's output. The standard variable types used in the Keithley User Library Tool (KULT) are:
  - **double:** Double precision data
  - **float:** Single precision, floating point data
  - **int:** Integer data
  - **long:** 32-bit integer data
  - **void:** No data returned
- **Apply**: Updates the module name and return type of the present library.
- **Library Hidden, Library Visible, Library UAP/Hidden, or Library UAP/Visible (one of these is displayed):** Indicates whether the presently open library is hidden or visible to test macros in the Keithley Interactive Test Tool (KITT) and available to user access points (UAPs) in Keithley Test Plan Manager (KTPM). For details, refer to Changing library attributes (on page 6-73).

Include area

The include area is located directly below the module identification area and displays header files and constants specified in the parameter box. The include area only displays information and cannot be accessed.

Module body area

The module body area is located below the include area and displays code of the active module for developing and editing purposes. Scroll bars located to the right and below the module body area let you move through the code.

Error and warning message area

This area contains any error and warning messages generated by the C language compiler. Selecting (double-clicking) a compile error listed in this window moves the cursor to the line that caused the error in the module body area. This facilitates error corrections. If no errors are generated during a compile, the following message is displayed: `No error/warnings reported, compilation/build was successful.`

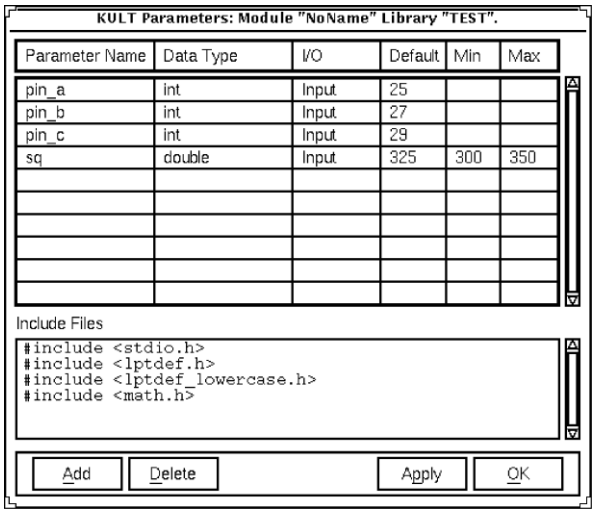
Parameter window

The Parameter window shown in the following figure is located at the top right side of the Keithley User Library Tool (KULT) screen and defines the variables specified in the module.

The Parameter window components are:

- Parameter definition area
- Include files area
- Control button area

Figure 63: Parameter window



Parameter definition area

The parameter definition area is the primary function of the Parameter window. The parameter definition area defines the Parameter Name, Data Type, and I/O field for all data included in the module call.

Parameter Name

The Parameter Name identifies the specified parameter.

Data Type

The Data Type specifies the parameter type. Clicking the arrow to the right of the Data Type field activates a pop-up menu listing the available data types, which are:

- **char:** Character data
- **char*:** Pointer to character data
- **float:** Single-precision, floating-point data
- **float*:** Pointer to single-precision, floating-point data
- **double:** Double-precision data
- **pointer:** Double-precision, floating-point data
- **int:** Integer data
- **int*:** Pointer to integer data
- **long:** 32-bit integer data
- **long*:** Pointer to 32-bit integer data
- **F_ARRAY_T:** Float array type
- **I_ARRAY_T:** Integer array type
- **D_ARRAY_T:** Double-precision array type

I/O field

The I/O field defines whether the parameter is an input or output type. Clicking the arrow to the right of the I/O field activates a pop-up menu that shows the input and output selections.

Default, min, and max fields

The following fields are used to specify default, minimum, and maximum parameters:

- **Default field:** For an input parameter, sets the default value for the specified parameter; for an output parameter, the default identifier name is set.
- **Min field:** Sets the suggested minimum value for the specified parameter, if applicable.
- **Max field:** Sets the suggested maximum value for the specified parameter, if applicable.

Include files area

The include files area is located at the bottom of the parameter box and lists the header files used within the module. This area can be used to add include files and #defines to the present module.

Control button area

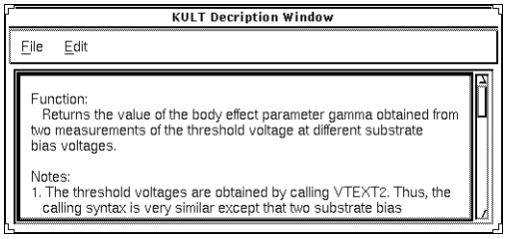
The control button area is aligned vertically along the right side of the parameter box and contains the following functions:

- **Add:** Adds a parameter template to the parameter definition area after the present parameter template.
- **Delete:** Removes a parameter template from the parameter definition area.
- **OK:** Updates the parameter list and header files of the module. The parameters and header files appear in the main window, and the Parameter window becomes hidden.
- **Apply:** Updates the constants and header files of the module. The parameter list and header files appear in the main window.

Description window

The Description window, shown in the following figure, is located at the bottom of the Keithley User Library Tool (KULT) screen. Information entered in this window is for module documentation and Keithley Interactive Test Tool (KITT) user library help. For more information about KITT, refer to Keithley Interactive Test Tool (on page 6-42).

Figure 64: Description window

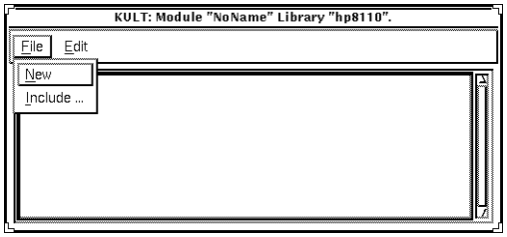


File menu

The File menu shown in the following figure has the following options:

- **New:** Clears the description area of text.
- **Include:** Lets you include an existing file in the description. Selecting Include produces the include dialog box. Use this box to locate the file you wish to include.

Figure 65: Description window file menu



Edit menu

The Edit menu, shown in the following figure, contains the following selections:

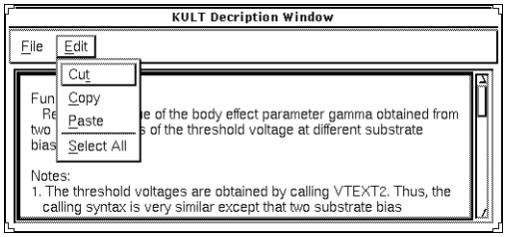
- **Cut:** Removes highlighted text, which can be restored to a different location using the Paste function.
- **Copy:** Copies highlighted text, which can be placed in a different location using the Paste function.
- **Paste:** Places cut or copied text at the cursor location.
- **Select All:** Lets you highlight all of the information entered in the description area for the purpose of cutting, copying, and pasting to a different location.

NOTE

When cutting, copying, and pasting between the main window and the Description window, remember to use the Cut or Copy command from the window that contains the text, and the Paste command from the window where the text is being placed.

You can also use the Edit commands that are accessed by clicking the middle mouse button, as described earlier in this section.

Figure 66: Description window edit menu



Description area

The description area is located below the menu bar and provides the area to enter help information.

NOTE

Do not place a period in the first column of any line in the description area. Any text after the period in the description area will not be available from the Keithley Interactive Test Tool (KITT) Command Help.

Command line interface

The Keithley User Library Tool (KULT) command-line interface lets you load, build, or delete libraries and add or delete modules from the command line. The format for command line instruction is:

```
kult subcommand [options] [filename...]
```

The subcommands are listed in the following table.

Command	Syntax	Description
new_lib	kult new_lib -l<*library_name*>	Start KULT and immediately load a new library from the command line.
bld_lib	kult bld_lib -l<*library_name*>	Start creating a new library immediately after starting KULT from the command line.
del_lib	kult del_lib -l<*library_name*>	Delete a library from the command line.
add_mod	kult add_mod-l<*library_name*> -d<*directory_name*> <*module*>	Add a module immediately after starting KULT from the command line. The -d option is needed when the modules you want to add are not in the present directory.
del_mod	kult del_mod -l<*library_name*> <*module*>	Delete a module from the command line.
compile_mod	kult compile_mod -l<*library_name*> <*module*>	Compile a module into the library *library_name*. The module must already exist.
update_mod	kult update_mod -l<*library_name*> <*module*>	Generates the prototype file, wrapper files, and help file for a module in the library *library_name*, then compiles the module. The module must already exist.

The options include:

- `-l<library_name>`
- `-d<directory_name>`
- `-hide`
- `-nohide`
- `-uap`
- `-nouap`
- `-dep [library_name].....[library_name]`

Example

The following is an example of how to use the command line to create a new library, add an existing module to the library, and then build the new library. This need can occur when you want to add a usrlib module from another facility or project to the current project.

NOTE

The new module name must be unique from any other module name in other user libraries.

In this example, you will be creating the new library, `mylib2`, and adding the module, `res3`. Both the old library and the new library are contained in the `/home/kthmgr` directory.

```
kult new_lib -lmylib2
kult add_mod -lmylib2 -d/home/kthmgr res3
kult bld_lib -lmylib2
```

If you want to add the module to an existing library, omit the kult `new_lib -lmylib2` step above. This assumes that the user library `mylib2` already exists and does not already contain a module named `res3`.

Copying user libraries with kult_copy_lib

The `kult_copy_lib` utility can be used to copy all or listed user library source files (`.c`) from the current or specified directory to a new user library in the current `KI_KULT_PATH`.

Usage:

```
kult_copy_lib -llibrary [-ddirectory [files]]
```

Where:

library = New user library name

directory = Directory path that contains the `.c` user library files

files = One or more files from the directory to include (default is `*.c`)

Example:

```
> ls *.c
   module1.c module2.c
   > kult_copy_lib -lmy_lib
   Library "my_lib" was created!
   module module1 added to library my_lib
   module module2 added to library my_lib
   Library my_lib built.
   done.
```

or

```
> ls ~/libfiles/*.c
   file1.c file2.c file3.c
   > kult_copy_lib -lmy_lib2 -d~/libfiles file1.c file3.c
   Library "my_lib2" was created!
   module file1 added to library my_lib2
   module file3 added to library my_lib2
   Library my_lib2 built.
   done.
```

Migrating user libraries with migrate_usrlib

The `migrate_usrlib` utility provides an easy method of copying usrlibs from previous Keithley Test Environment (KTE) installation packages to the current KTE installation package when the current package is installed in an alternate directory. It can be used to copy any usrlib from any project of any previously installed package into the current `KI_KULT_PATH` directory.

The `migrate_usrlib` utility first looks for previously installed packages, then prompts you to select the one where the source usrlib is located. If the `-p` option is used, it then displays a list of projects associated with the package, and prompts you to select one. If `-p` is not used, or if no projects (other than the default) are associated with the selected package, the script assumes that the default project contains the source usrlib. It then displays a list of usrlibs in the selected package or project, and prompts you to select one (or all) to copy. It proceeds to call `kult_copy_lib` to copy the selected usrlibs into the `KI_KULT_PATH` directory, and then exits.

Usage:

```
migrate_usrlib [-p]
```

Migrating from S400 test plans to S540

The S540 Keithley Test Environment (KTE) software is mostly compatible with existing S400 KTE test plans. However, due to operating system and instrumentation differences, some minor changes are necessary for your measurement routines. This section identifies some of the most common items.

1. S540 KTE software executes in the CentOS Linux® environment and uses the GCC Compiler. The GCC Compiler may generate more warnings than the Sun Solaris Workshop compiler tools. Adding `#include <stdlib.h>` and `#include <string.h>` lines to your Keithley User Test Library (KULT) modules may be enough to resolve the warnings. Please note that these are compiler warnings only. The modules will still execute, but it is always a good practice to have your usrlib module code compile without warnings.

NOTE

If you are using the Recipe Manager/Version Control feature of KTE, you must have your usrlib modules compile without warnings.

2. The S540 uses SMUs, not VIMS. As a result, any references to VIMS need to be changed to SMU.
3. The S540 LPTLib library returns data as "double" instead of "float." Your usrlib modules may need to be modified to support double data types instead of float.
4. The S540 system has all of the source-measure unit (SMU) low terminals connected to each other. As a result, you cannot stack two or more SMUs together to increase the voltage. Your usrlib modules may need to be modified to remove the use of this stacking capability.
5. The polarity value of current measurements for the S540 SMUs is reversed from the S400 VIMS. As a result, any use of the `trigX` commands will need to be changed. The current measurement polarity of the S540 SMUs and S600 SMUs are the same, so no change is necessary.
6. Any OS-specific user access point (UAP) code and user scripts may need to be ported to Linux® from Solaris. Most UAP routines will need to be recompiled in the new S540 KTE environment.

Locking a module

Each time a module is opened in the Keithley User Library Tool (KULT), a lock file is created at `$KI_KULT_PATH/"libname"/lock` that prevents other users from modifying the module while it is open. The module may be accessed and used, but no changes can be made to the module without changing the name of the file.

When an attempt is made to access a module that is already open, a dialog box is displayed stating that the module selected is locked and the present user's name and PID.

After clicking the **OK** button, the module opens normally. If any changes are made to the module while it is locked by another user, this edited module must be given a new name. This new name cannot be a name that is already being used, or KULT will not allow the module to be saved.

WARNING

If a library is deleted while another user is working on a module from that library, the module will still be deleted without warning, even though it is locked, and the user will be prompted that the file no longer exists when a Save is attempted. The user should then save the file to a new module.

NOTE

It is a good practice to periodically check the home directory of the lock files and clean up any files that no longer require locks.

When a module is opened, a lock file is created at `$KI_KULT_PATH/"libname"/lock` in the following manner:

- The lock file is named *X* `-` *Y* `.lck`, where *X* is the library name and *Y* is the module name.
- The following information is contained in the lockfile:

  PID:**** The processor ID

  USER:**** User name

  HOST:**** Host computer name

  TIME:**** Time lock was created

  LIB:**** Library name

  MOD:**** Module name

  FILE:**** Path/Lock file name

Changing library attributes

You can change library attributes; the following topics describe how.

Hiding libraries from Keithley Interactive Test Tool macros

A library that you create may contain modules that will not work if called by Keithley Interactive Test Tool (KITT) macros (for example, modules that are designed only to be called by other modules in other user libraries). The Hide Library menu item and the hide command-line option allow you to make a library unavailable to macros by hiding the library from KITT.

Application example

The following example assumes user library `LibB` contains low-level modules that only modules in user library `LibA` can call. By setting the dependency of `LibA` on `LibB` and then hiding `LibB`, only the modules in `LibA` will be able to call the modules in the hidden `LibB`.

Procedures

To hide a user library from the Keithley Interactive Test Tool (KITT), use either of the following methods.

To hide a user library using the KULT window:

1. Open the library in KULT.
2. From the **Options** menu, select the **Hide Library** toggle. The Library Visible or Library UAP/Visible indication located in the upper right of the KULT main window changes to Library Hidden or Library UAP/Hidden.
3. In the **Options** menu, select **Build Library** to rebuild the library as a hidden library.

Or:

Using the command line, enter the following:

```
kult bld_lib -l<library_name> -hide
```

To make a user library visible in KITT, use either of the following methods.

To make a user library visible using the KULT window:

1. Open the library in KULT.
2. From the **Options** menu, deselect the **Hide Library** toggle. The Library Hidden or Library UAP/Hidden indication located in the upper right of the KULT main window changes to Library Visible or Library UAP/Visible.
3. In the **Options** menu, select **Build Library** to rebuild the library as a library that is hidden from KITT.

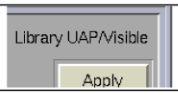
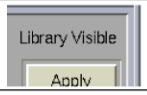
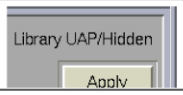
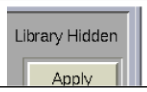
Or:

Using the command line, enter the following:

```
kult bld_lib -l<library_name> -nohide
```

The following table summarizes Hide Library use combined with UAP Library use (discussed in the topics that follow).

Figure 67: Effect of library options on library availability to macros and UAPs

Combination of options that is set at the command line or in the Options menu		Library state		Indication that displays above the Apply button
bld_lib command options[2]	Options menu selections[2]	Available to test macros and visible in KITT	Available at UAPs and in KTPM[3]	
-nohide[1] -uap[1]		●	●	
-nohide[1] -nouap		●	No	
-hide -uap[1]		No	●	
-hide -nouap		No	No	

Setting library availability at UAPs

Some libraries that you create may contain modules that do not work in user access points (UAPs). The UAP Library toggle, when deselected, and the `-nouap` command-line option make these libraries unavailable at UAPs and in the Keithley Test Plan Manager (KTPM). Refer to the procedures below and the previous table.

To make a user library unavailable at UAPs and in KTPM, use either of the following methods.

To make a user library unavailable in the KULT window:

1. Open the library in KULT.
2. From the **Options** menu, deselect the **UAP Library** toggle (UAP Library is selected by default). The Library UAP/Visible or Library UAP/Hidden indication located in the upper right of the KULT main window changes to Library Visible or Library Hidden.
3. In the Options menu, select "Build Library" to rebuild the library.

Using the command line, enter the following:

```
kult bld_lib -l<library_name> -nouap
```

To make a user library available at UAPs and in KTPM, use either of the following methods.

To make a user library available using the KULT window:

1. Open the library in KULT.
2. From the **Options** menu, select the **UAP Library** toggle. The Library Visible or Library Hidden indication located in the upper right of the KULT main window changes to Library UAP/Visible or Library UAP/Hidden.
3. In the **Options** menu, select **Build Library** to rebuild the library.

Using the command line, enter the following:

`kult bld_lib -l<library_name> -uap`

KTE library locking

The following topics describe Keithley Test Environment (KTE) library locking.

Run-time library locking

You can lock a run-time library using the following procedure.

LIBRARY reader locks for KTXE, KSOX, and KITT

LIBRARY reader locks for Keithley Test Execution Engine (KTXE), Keithley Sequential Executor (KSOX), and Keithley Interactive Test Tool (KITT). These locks are set when a process opens the library for use. These locks reside in the LIBRARY lock directory (`$KI_KULT_PATH/lock`). The lock file is named:

```
libName.pid.hostname
```

Where:

`libName` = The name of the library
`pid` = The pid of the process that is using the library
`hostname` = The hostname of the machine using the library

The following programs and tools use the library reader locks:

Keithley Sequential Executor (KSOX)

When you attempt to open a usrlib, check for the existence of a build lock. If a build lock exists for this library, skip the `dlopen` call. No reader lock is created because the `dlopen` was not called. Delete the reader lock when a library is closed.

Keithley Interactive Test Tool (KITT)

KITT only opens the usrlib when it needs references for syntax checking or for loading the command list.

KITT uses the KSOX utilities to create and delete the reader locks. The same rule applies if a build lock exists while attempting to open a library; the open will fail.

KTXE

KTXE will use KSOX utilities to create and delete the reader locks. The same rule applies if a build lock exists while attempting to open a library; the open will fail.

Edit-time library locking

You can lock an edit-time library using the following procedure.

LIBRARY write lock for the Keithley User Library Tool

This lock is created and exists only while the Keithley User Library Tool (KULT) is building the library. This lock resides in the LIBRARY lock directory (`$KI_KULT_PATH/lock`). The lock file is named:

`libName`

Where:

`libName` = The name of the library

KULT checks for the presence of any reader locks before attempting a build library operation. If a read lock exists, a warning message is displayed and the library build does not occur.

KULT creates a write lock before the build operation starts. This write lock is removed when the build operation has completed.

KULT uses the existing module lock scheme.

KULT uses the existing module lock scheme to prevent multiple edits of the same module. The location of these lock files changed from the old lock directory, `$KIHOME/lock`, to a new location:

`$KI_KULT_PATH/libName/lock`

Where:

`libName` = The name of the library.

Troubleshooting

Occasionally, an error occurs and lock files are not deleted or removed by the appropriate application. The following are some situations that can cause problems:

- **Using the debugger and not running the program to completion.** Using the debugger to test a program and aborting the session before the program can complete normally leaves reader locks present in the `$KI_KULT_PATH/lock` directory.
- **Using the `kill -9` command.** If a process is stopped using the `kill -9` command, reader locks are not removed automatically from the `$KI_KULT_PATH/lock` directory.

Deleting the lock files resolves the above problems.

Limits File Editor (LFE)

The Limits File Editor (LFE) creates and edits limits files (`.klf`). Spreadsheet and Dialog views are provided for data manipulation. The Dialog view is restricted to a single parameter at a time. The following information is included in the file for every test parameter:

Character fields: ID, Name, Units, Category

Choice fields: Report, Critical, Abort Action, Abort Limit, and Enabled

Data fields: Target, four limit pairs (Valid, Spec, Ctrl, and Engr), Class, and three user data fields (User 1, User 2, and User 3).

LFE data is used both during test execution and after test execution. Abort flags can be set to interrupt testing if preliminary test results on a few parameters include gross errors. In such cases, testing can be delayed for operator intervention, corrective action, and retry. User access point (UAP) code can use LFE data to perform simple or complex run-time actions.

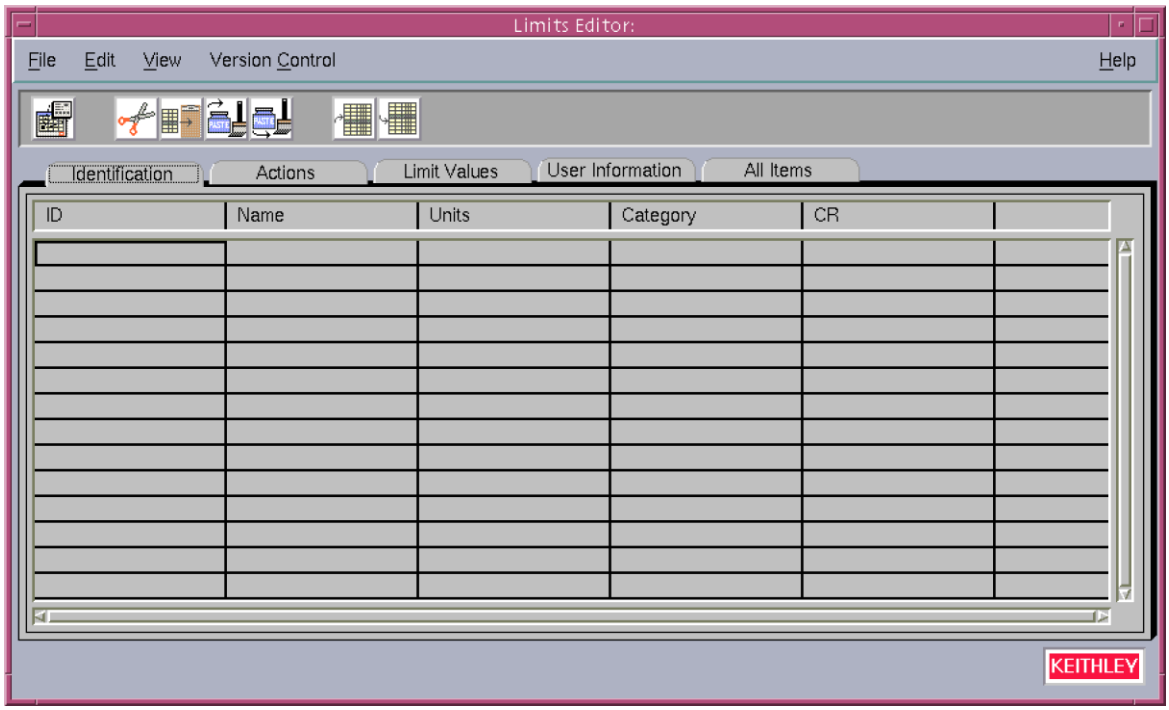
LFE data is also used by the Keithley Summary Utility (KSU) and the Lot Summary program to create reports. These reports show the acquired data compared against the Valid, Spec, Ctrl, and Engr limit pairs. The limits file used during production testing is specified in the Wafer Plan.

LFE main window

The Limits File Editor (LFE) main window, shown in the following figure, is the primary interface of LFE.

LFE is a full-screen graphical application with a spreadsheet format. The main window contains four pull-down menus, seven push buttons that correspond to menu selections, and a data entry area organized by five tabs.

Figure 68: LFE main window



Title bar

The title bar lists the name of the currently open limits file and the read-write status of the file.

The pull-down menus contain the following functions:

- Select and save parameter limits files
- Exit the Limits File Editor (LFE)
- Manipulate data in a limits file
- View limits file information in a different format
- Access help

The following paragraphs contain a brief overview of each menu.

LFE File menu

You can use the File menu to do the following tasks:

- Reinitialize the main window (new)
- Load or save limits files
- Delete limit files
- Generate limits from Keithley Test Macros (KTM) files
- Set a limit as the default limit
- Save column settings
- Exit the Limits File Editor (LFE)

LFE Edit menu

You can use the Edit menu to manipulate limits by cutting, copying, pasting, and inserting. You can also do a search for specific data (character strings) or sort the data by the selected data field.

LFE View menu

You can use the View menu to access the Limits Editor Dialog window and enter a comment line that will be stored with the file.

LFE Help menu

You can use the Help menu to acquire online help information about the Limits File Editor (LFE) and the release version of the LFE being run.

Push-buttons

Seven push-buttons located beneath the menu bar provide shortcuts to menu selections.

Limits data entry area

The limits data associated with each parameter is arranged along one row. Multiple rows contain the same limits data for multiple parameters. The information is organized by tabs, allowing you to quickly find and modify the fields:

- **Identification:** Contains the ID, Name, Units, Category, and Critical fields.
- **Actions:** Contains the ID, Name, Enabled, Class, Report, Abort Action, and Abort Limit fields.
- **Limit Values:** Contains the ID, Name, Target, Valid (H/L), Spec (H/L), Control (H/L), and Engineering (H/L) fields.
- **User Information:** Contains the ID, Name, User 1, User 2, and User 3 fields.
- **All Items:** Contains all the fields.

ID

The ID uniquely identifies each test parameter. This entry must be a legal C language name. The string may contain a maximum of 128 characters. This field is required.

Name

The name field provides for a descriptive name that describes the parameter more clearly than the ID. The data in this field is used by the Keithley Summary Utility (KSU). This field may contain non-alphanumeric characters (including spaces) and need not be a legal C language name. The only constraint is that it must not include commas. The string can contain a maximum of 40 characters.

Units

The Units field defines the units for the limit's test results. This field may contain non-alphanumeric characters (including spaces) and need not be a legal C language name. The only constraint is that it must not include commas. The string can contain a maximum of 10 characters. Examples of entries for this field are `volts` or `amps`.

Category

The Category field allows you to specify what group the results data will belong to when all of the results data is sorted into database groups. Each category must be a single-word string. One parameter can belong to several categories if all the categories are listed (comma- or white space-delimited). This field can contain a maximum of 20 characters.

RP

RP (Report) determines whether or not the limit is included in the lot summary report compiled in the Keithley Summary Utility (KSU). The entry for this field is `Y` for yes and `N` for no.

CR

CR (Critical) allows parameters to be tagged as not critical (`N`) or as critical with one of nine different critical flags (`1`, `2`, `3`, `4`, `5`, `6`, `7`, `8`, `9`).

Target

The Target field identifies the ideal result expected for the specified result ID. The entry for this field is a numeric value.

Abort Action

The Abort Action field identifies the sequencer execution engine level to proceed to if the measured result fails the abort limit check. This field toggles between the following values: Subsite, Site, Wafer, Lot, or None.

Abort Limit

The Abort Limit field specifies the limits to use for the Abort Action results comparison in the execution engine. If an abort flag is triggered by the measured value, the sequencer loops to the sequencer level specified by the Abort Action field. This field toggles between the following values: Valid, Spec, Ctrl, or Engr.

Valid, Spec, Ctrl, and Engr high and low parameters

Each limit has up to four different high and low limits that can be set. These limits can be used during test program execution to establish testing abort criteria. These limits are used to sort parameter test results in the Keithley Summary Utility (KSU) lot summary report. These ranges can be set to any user-specified parameters and are expressed in scientific notation. Following are four typical applications for the parameters.

- **Valid:** This limit pair can be used to check for faults within the testing system. Numbers outside of this range indicate that the data being produced by the system is invalid, usually due to an equipment malfunction.
- **Spec:** This limit pair can be used to check for manufacturing standards. Numbers outside of this range indicate that the device being tested does not meet manufacturing specifications.
- **Ctrl:** This limit pair can be used to check process control limits.
- **Engr:** This limit pair can be used by engineers to ensure the components meet design standards.

Enabled

The Enabled field is used only with the Adaptive Test option software. This field determines whether this parameter is included in the test. The entry for this field is `Y` for yes and `N` for no.

Class

The Class field can be used with the enabled flag or as user data. This field can contain a maximum of 255 characters.

User fields

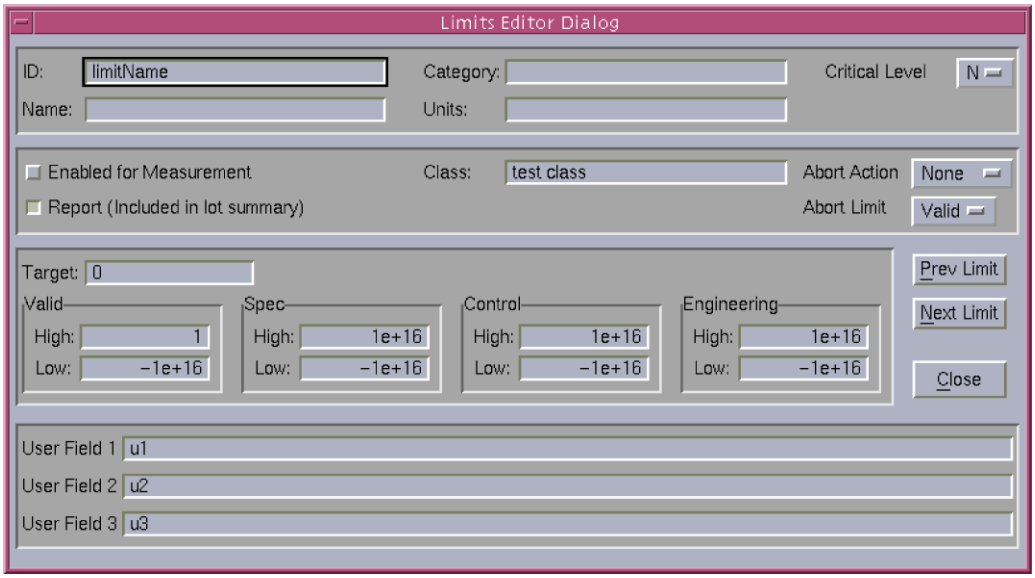
The User fields can contain a maximum of 255 characters of information. There are three user data fields:

- User Field 1
- User Field 2
- User Field 3

Limits Editor Dialog window

The Limits Editor Dialog window displays all of the elements for a parameter, one parameter at a time, as shown in the following figure. A single parameter in the spreadsheet format is one row. The parameter that appears in this window is the parameter that is highlighted in the main window.

Figure 69: Limits editor dialog window



This window also tracks the movement of the cursor in the Limits File Editor (LFE) main window. As changes are made in the main window, the information in the Limits Editor Dialog window is automatically updated.

There are two check boxes in this window for the Report (RP) and Enabled columns in the main window. Checking these boxes is the same as entering `Y` in the corresponding box in the main window. There are choice boxes for the Critical (CR), Abort Action, and Abort Limit fields, allowing you to select one of the available options for each field.

Keithley Test Plan Manager (KTPM)

The following paragraphs contain basic information about the Keithley Test Plan Manager (KTPM). Brief descriptions of the Wafer Plan Builder, the Cassette Plan Builder, and the Test Documentation Tool are included.

KTPM is a graphical user interface used by the test engineer to create a cassette plan file (`.cpf`) that can be executed using the Keithley Test Execution Engine (KTXE). The information that is entered into the `.cpf` file determines the parameters of the test process that will be executed.

Wafer plan builder

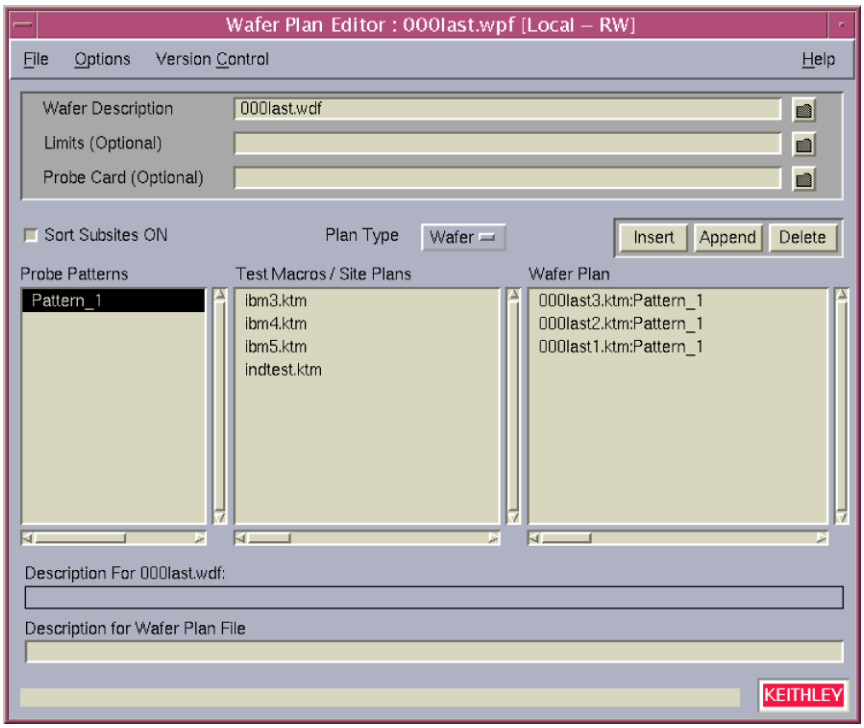
The Wafer Plan Builder is used to create a wafer-testing plan. The most important function performed in the wafer plan builder is the binding of tests to wafer probe patterns. This can be simplified for many wafer plans by using the feature of grouping test macros into site plans.

Wafer plan builder main window

The wafer plan builder window shown in the following figure contains the following areas:

- **Main menu bar:** Contains the File, Options, and Help menus.
- **Wafer description file selection field:** Used to select the wafer description file.
- **Limits file selection field:** Used to select the limits file.
- **Probe card file selection field:** Used to select the probe card file.
- **Sort Subsites button:** Used to modify the subsite probing sequence.
- **Plan type selection button:** Used to select between building a wafer plan or a site plan.
- **Probe Patterns/Site Plan list:** Gives a listing of all of the probe patterns or site plans available.
- **Test Macros/Site Plans list:** Gives a listing of all the valid test macros and site plans for the wafer description file specified.
- **Site plan/Wafer plan builder field:** For site plans, this field contains the macros in the site plan. For wafer plans, this field contains the macros and site plans bound to the probe pattern.
- **Insert buttons:** Used to place selected files before or after a file in the builder field, or to delete a file from the builder field.
- **New site plan button (available only when the plan type button is set to Site Plan):** Opens a dialog box that allows you to enter the name and description of a new site plan.
- **Wafer plan file description field:** Lets you enter a description of the wafer plan file.
- **Wafer description file description field:** Displays the description of the selected `.wdf` file. This field cannot be edited.

Figure 70: Wafer plan builder



Title bar

The title bar lists the name of the currently open wafer plan file and the read-write status of the file.

Wafer plan builder File menu

The File menu contains the following selections:

- **New:** Clears the Wafer Plan Builder main screen to prepare for creating a new wafer plan file.
- **Open:** Opens an existing wafer plan file.
- **Save:** Saves the currently displayed wafer plan file under the current name.
- **Save As:** Saves the currently displayed wafer plan file under a new or pre-existing user-specified name.
- **Delete:** Opens a window allowing you to delete a wafer plan file.
- **Exit:** Exits the wafer plan builder.

Wafer plan builder Options menu

The Options menu contains the toggle Save Default Path. This toggle enables or disables the storing of the filenames within the `.wpf` file. The default environment variables are assumed and used for file locations. If this toggle is disabled and the user selects a file that is not in the default location, the path is saved with the filename. If the toggle is enabled, the path is always saved in the `.wpf` file.

Wafer plan builder Help menu

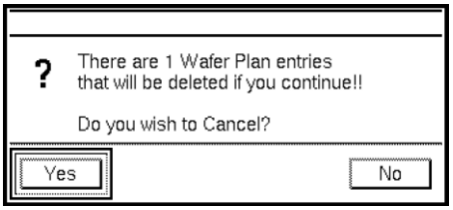
The Help menu provides online help and release version information about the wafer plan builder. The Documentation option allows access to the online Keithley Test Plan Manager (KTPM) manual. The About option displays the software revision information.

Wafer description file selection field

The wafer description file selection field lets you select the `.wdf` file that will be used during test execution. This is an optional field overrides the `.wdf` file specified in any wafer plan file used. Right-clicking the filename in this field displays a pop-up dialog box allowing you to open the wafer description file for editing.

If a test macro site plan has been entered into the wafer plan and the `.wdf` folder button is selected, the warning box shown in the following figure is displayed, telling you that there are wafer plan entries that will be deleted if you continue selecting a new `.wdf` file. Select **Yes** to cancel the procedure or **No** to continue.

Figure 71: Wafer plan entry warning box



Limits file selection field

Clicking the limits file selection field folder icon lets you select the `.klf` file that will be used during wafer plan execution. Right-clicking the filename in this field displays a pop-up dialog box allowing you to open the limits file for editing.

Probe card file selection field

The probe card file selection field lets you select the `.pcf` file that will be used during test execution. This is an optional field that overrides the `.pcf` file specified in any wafer plan file used. Right-clicking the filename in this field displays a pop-up dialog box allowing you to open the probe card file for editing.

Plan type selection button

Click and hold on this button to choose between building a Wafer Plan and building a Site Plan.

Sort subsites button

When this button is selected, the subsites are sorted for testing by their increasing X-axis and Y-axis values. When this button is not selected, the subsites are tested in the order they appear on the subsite list.

Probe patterns and site plans list

When the plan type selection button is set to Wafer Plan, this area contains a list of all the probe patterns available from the wafer description file selected. When Site Plan is selected, this area lists all of the site plans available for use when building the wafer plan. Site plans are saved within each `.wpf` file only; there is no master list of site plans accumulated across `.wpf` files.

Test macros and site plans list

When the plan type selection button is set to Wafer Plan, this area contains a list of all the test macros and site plans available. When Site Plan is selected, this area lists all of the test macros available for use when building a site plan.

Site plan and wafer plan builder field

This field is used to build the wafer plan or site plan. By clicking the different files in the Probe Pattern/Site Plan list and the Test Macros/Site Plan list, and then clicking the Insert Bef. or Insert Aft. buttons, you can add the highlighted files to the site plan or wafer plan you are building. When macros or site plans are bound to probe patterns, those macros or site plans are removed from the list of available macros and site plans. Each macro or site plan can be bound to only one probe pattern.

Insert buttons

Used to move the files from the Probe Pattern/Site Plan list and the Test Macros/Site Plan list into the Site Plan/Wafer Plan Builder field. The function of each of these buttons is:

- **Insert button**: Inserts the highlighted files from the Probe Pattern/Site Plan list and the Test Macros/Site Plan list before the highlighted file in the Site Plan/Wafer Plan Builder field.
- **Append button**: Inserts the highlighted files from the Probe Pattern/Site Plan list and the Test Macros/Site Plan list after the highlighted file in the Site Plan/Wafer Plan Builder field.
- **Delete button**: Removes the file highlighted in the Site Plan/Wafer Plan Builder field.

New site plan button

This button appears when the plan type selection button is set to Site Plan. Clicking this button opens a dialog box that lets you enter the name and description of a new site plan. Once a site plan description is entered and a site plan is formed, it cannot be viewed or edited in the wafer plan editor, and will only appear in the `.wpf` file whenever the site plan is used in that `.wpf`. This new site plan appears in the Site Plan list when the plan type selector button is set to Site Plan, and in the Test Macros/Site Plans list when the Plan Type selector button is set to Wafer Plan.

Wafer plan file description field

This field lets you enter a description of the wafer plan file, which can be edited and saved.

Wafer description file description field

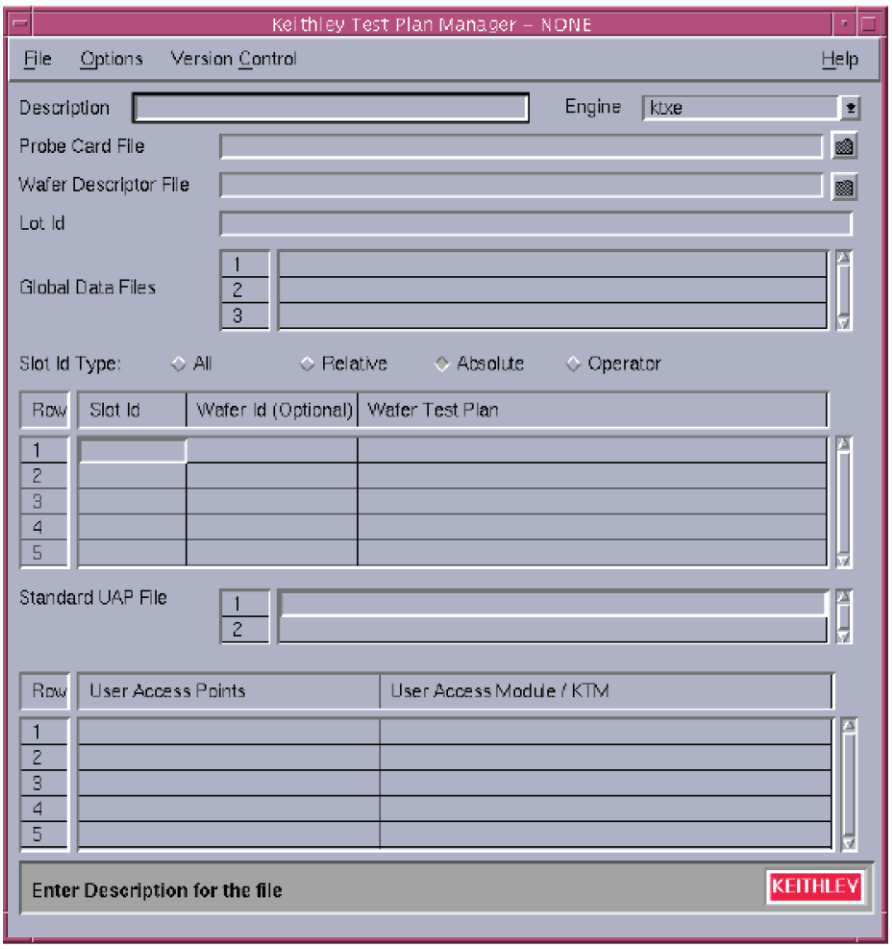
Displays the description of the selected `.wdf` file. This field cannot be edited.

Cassette plan builder

The cassette plan builder, shown in the following figure contains the following areas:

- **Main menu bar:** Contains the File, Options, and Help menus.
- **File description field**: Enter a description of the cassette plan file.
- **Execution engine selection field:** Select a specific execution engine.
- **Probe card file selection field:** Select the probe card file to be used for all wafers in the cassette.
- **Wafer description file selection field:** Select the wafer description file to be used for all wafers in the cassette.
- **Lot ID data field:** Enter the ID of the lot that is to be tested.
- **Global data file selection field:** Enter the files containing all the global data definitions.
- **Wafer test plan area:** Identify the Wafer Test Plan to be used for each wafer during testing.
- **Standard UAP file field:** Identify the files containing any standard user routines to be added to the execution engine. If multiple files are identified they are executed in the order entered.
- **UAP Module/KTM area:** Identify any additional modules or Keithley Test Modules (KTMs) to be added to the execution engine at the specified user access point (UAP).

Figure 72: Cassette plan builder



Title bar

The title bar lists the name of the currently open cassette plan file and the read-write status of the file.

Cassette plan builder File menu

The File menu contains the following selections:

- **New:** Clears the cassette plan builder main screen to create a new cassette plan file.
- **Open:** Opens an existing cassette plan file.
- **Save:** Saves the currently displayed cassette plan file under the current name.
- **Save As:** Saves the currently displayed cassette plan file under a new or pre-existing user-specified name.
- **Delete:** Opens a window allowing you to delete a cassette plan file.
- **Exit:** Exits Keithley Test Plan Manager (KTPM).

Cassette plan builder Options menu

The Options menu contains the following selections:

- **Wafer Plan Editor:** Opens the wafer plan editor window, allowing you to create or modify a wafer plan.
- **Test Documentation Tool:** Opens the Test Documentation Tool, allowing you to view the cassette plan as it will be executed by the selected execution engine.
- **Save Default Path:** This toggle enables or disables the storing of the various filenames within the `.cpf` file. The default environment variables are assumed and used for file locations. If this toggle is disabled and you select a file that is not in the default location, the path is saved with the filename. If the toggle is enabled, the path is always saved in the `.cpf` file.

Cassette plan builder Help menu

The Help menu provides online help and release version information about Keithley Test Plan Manager (KTPM). The Documentation option allows access to the online KTPM manual. The About option displays the software revision information.

File Description field

The file Description field is used to enter a short description about the cassette plan file.

Execution Engine selection field

The execution Engine selection field lets you select the execution engine that will run the cassette plan.

Probe card file selection field

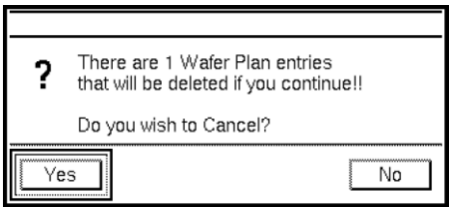
The probe card file selection field lets you select the `.pcf` file that will be used during test execution. This is an optional field that overrides the `.pcf` file specified in any wafer plan file used. Right-clicking the filename in this field displays a pop-up dialog box allowing you to open the probe card file for editing.

Wafer description file selection field

The wafer description file selection field lets you select the `.wdf` file that will be used during test execution. This is an optional field overrides the `.wdf` file specified in any wafer plan file used. Right-clicking the filename in this field displays a pop-up dialog box allowing you to open the wafer description file for editing.

If a test macro site plan has been entered into the wafer plan and the `.wdf` folder button is selected, the warning box shown in the following figure is displayed, telling you that there are wafer plan entries that will be deleted if you continue selecting a new `.wdf` file. Select **Yes** to cancel the procedure or **No** to continue.

Figure 73: Wafer plan entry warning box



Lot ID data field

This is an optional field that lets you identify the lot that will be tested. This data is used as the default if none is specified at run time by the operator or shop floor control system.

Global data file selection field

The global data file selection field is an optional field that lets you specify any `.gdf` files that will used to define data during plan execution. The full path name of the file should be given.

Slot ID type selection buttons

These buttons apply to the Slot ID column in the wafer test plan. Four different selections are possible:

- **All:** All wafers within a cassette will be tested with the same wafer plan.
- **Relative:** Each wafer should be identified by its slot position relative to the other wafers in the cassette.
- **Absolute:** The wafer should be identified by its actual slot position in the cassette.
- **Operator:** Allows the operator to select which wafers to test.

Wafer test plan field

The wafer test plan field lets you specify the wafer test plan for a specific wafer. This area lets you specify the slot that contains the wafer, the wafer ID (optional), and the path and filename of the wafer test plan.

Standard user access point file field

The Standard UAP file field is optional and lets you specify files that contain module and Keithley Test Module (KTM) routines to be used at user access points (UAPs) in the execution engine.

User access point module and Keithley test module area

The UAP Module/KTM area lets you specify the module or `.ktm` file that will be executed at the user access point (UAP) listed. Available UAPs are shown by clicking the right mouse button in this field and selecting Add New UAP After or Insert New UAP. A list arrow is displayed that lets you select from all the UAPs available to the selected execution engine. Each engine will have a different list of available UAPs. The UAP list and descriptions are stored in the `ktpm.ini` file.

Test documentation tool

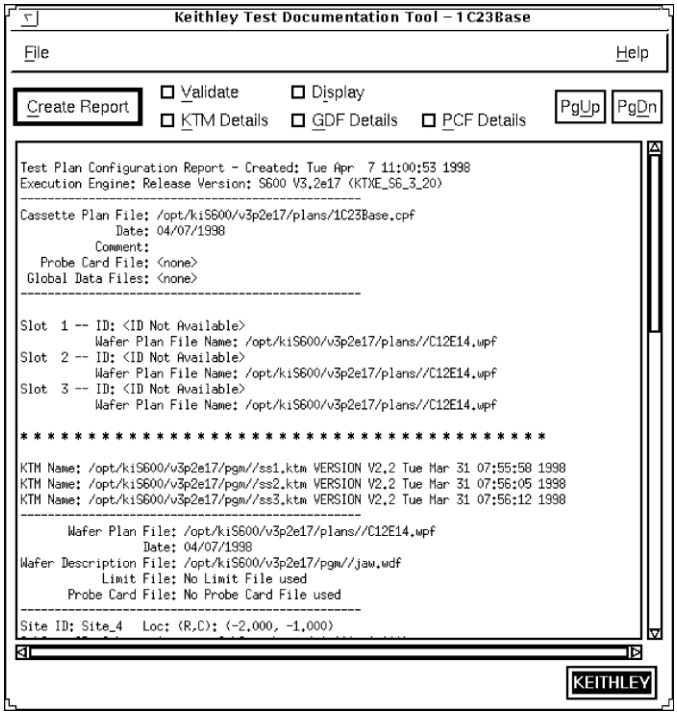
The test documentation tool is used to view the test that will be performed by the cassette plan file specified in the Keithley Test Plan Manager (KTPM). The tool provides a report that shows each user access point (UAP) and `.ktm` file that is executed, in the order of execution for all wafers, sites, and subsites.

Main window

The test documentation tool main window, shown in the following figure, contains the following areas:

- **Main menu bar:** Contains the File and Help menus.
- **Create report button:** Regenerates the report with any new options specified.
- **KTM details toggle:** Enables or disables the display of Keithley Test Module (KTM) details.
- **GDF details toggle:** Enables or disables the display of Global Data File details
- **PCF details toggle:** Enables or disables the display of Probe Card File details.
- **Validate toggle:** Enables or disables additional cassette plan validation steps.
- **Display toggle:** Enables or disables displaying the test plan sequence report.
- **PgUp/PgDn buttons:** Lets you scroll through the report page-by-page rather than line-by-line.
- **Report field:** Shows the report created by the Test Documentation Tool.

Figure 74: Test documentation tool



File menu

The File menu contains the following selections:

- **Save As:** Saves the information to a new file.
- **Print:** Provides a complete printout of the report.
- **Exit:** Exits the Test Documentation Tool.

Help menu

The Help menu provides online help and release version information about the Test Documentation Tool. The Documentation option allows access to the online Keithley Test Documentation Tool (KTDT) manual. The About option displays the software revision information.

Create report button

When this button is clicked, the report is created with either the Keithley test module (KTM), global data file (GDF), and probe card file (PCF) Details enabled or disabled.

Keithley test module Details toggle

This toggle button is used to enable or disable the display of all Keithley test module (KTM) details. These details consist of the actual test commands within the KTM. Enabling this option could cause the report to be very large. This option provides a way to create a detailed printout for archiving purposes.

Global data file Details toggle

This toggle button is used to enable or disable the display of the global data file (GDF) details. These details consist of the actual global data names and values. Enabling this option could cause the report to be very large. This option provides a way to create a detailed printout for archiving purposes.

Probe card file Details toggle

This toggle button is used to enable or disable the display of the probe card file (PCF) details. These details consist of the actual pin names and assignments. Enabling this option could cause the report to be very large. This option provides a way to create a detailed printout for archiving purposes.

Validate toggle

This toggle button is used to enable or disable additional cassette plan validation steps. These additional steps include checking the path names and file locations of all the files associated with the selected cassette plan. Any errors that are identified are logged to the `$KI_KTXE_CPF/cassPlanName.cpf.err` file. These errors are also shown in the documentation window.

Display toggle

This toggle button is used to enable or disable displaying the test plan sequence report after the report has been created. A printout of the report can be created even if the report is not displayed on screen.

PgUp and PgDn buttons

These buttons allow you to scroll through the report page by page. To move through the report at smaller intervals, use the scroll bar along the right side of the report field.

Report field

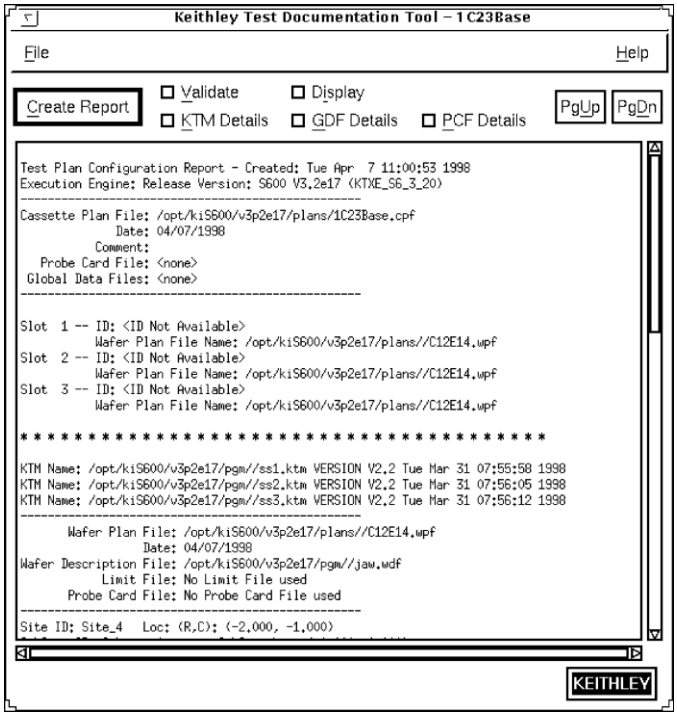
This field is where the entire test report can be viewed. If a wafer plan file has been specified for multiple slots, only the full details of the first occurrence of the wafer plan file is shown.

Using the Test Documentation Tool

Once you have completed building your cassette plan file, you can preview the actual test plan using the Test Documentation Tool.

Select **Test Documentation Tool** from the **Options** menu. The Test Documentation Tool is displayed, as shown in the following figure.

Figure 75: Test documentation tool



Enable or disable the appropriate options for your report and press the **Create Report** button. The system creates and displays the execution report in the display window. From this window you can review your cassette plan to ensure that the test plan will perform the tests in the proper sequence.

For all wafers, sites, and subsites, you can also examine each of the test macros by clicking the Keithley Test Manager (KTM) **Details** toggle button and then the **Create Report** button. You can also print out a copy of the entire test plan by selecting **Print** from the **File** menu. This will give you a printed copy of the test plan that can be stored in your archives. The **Save As** option from the **File** menu enables you to save an electronic copy of the report for archival.

Generating documentation during testing

It is possible to generate documentation during Keithley Test Execution Engine (KTXE) execution with the addition of a command-line switch to KTXE. The switch used is `-doc DOC_ON`. After KTXE completes execution, a file is placed in the `$KITMP` directory with the same base name as the cassette plan with the extension `.cpf.kxd`. The utility CreateExecDoc should then be executed to produce a report. The report is written to the `$KITMP` directory with the cassette plan base name `.cpf.kxd.txt`. For more information on CreateExecDoc, enter `CreateExecDoc -help`.

Usage:

```
ktxe -cpf cassette_plan -doc DOC_ON
   CreateExecDoc -cpf cassette_plan
```

Example:

```
> ktxe -cpf myplan.cpf -doc DOC_ON
   > CreateExecDoc -cpf myplan.cpf
   > ls myplan.*
   myplan.cpf myplan.cpf.kxd myplan.cpf.kxd.txt
```

Adaptive testing

Adaptive testing is an optional component of the Keithley Test Environment (KTE) that enables the test plan to change for each wafer being tested. There are two main components of adaptive testing: Zone-based testing and result-based testing. Zone-based testing creates random site test patterns, and result-based testing changes the sites or tests to be used based on the results of previous site tests. Both of these methods are described below.

Zone-based testing

Zone-based testing is a test method where the sites to be tested are generated at run time. The sites to be tested are selected randomly from predefined zones, or patterns, contained in the wafer definition file. For information about creating patterns in the wafer definition file, refer to Wafer Description Utility (on page 6-15).

Zone-based test modes

Zone-based testing can be initiated using two different methods. The first method, described below, is a command line switch for use with the Keithley Test Execution Engine (KTXE). The second method, described under Global data variables (on page 6-99), is with the use of the `KTXE_RP_test_mode` variable. There are four modes available when using zone-based testing. Testing modes are selected from the command line by specifying the `-u "X"` argument when executing KTXE, where "*X*" is the test mode. For example:

```
ktxe -cpf MyPlan.cpf -u "A" -w TestZones.wdf
```

The test modes are as follows:

- **N – Normal Test:** When normal testing is selected, no patterns are generated at run time. This is the default behavior of KTXE. If the `-u` command line argument is not specified when executing KTXE, this mode is used.
- **R – Runtime:** This test mode generates random patterns based on the initial wafer description file and the frequency of sites on each wafer. The frequency of sites per wafer is defined in global data by `KTXE_RP_max_freq`. The initial wafer pattern defines zones for testing. Each zone is defined by a pattern. Each pattern is associated with a wafer plan file generated at run time from an initial wafer plan file.
- **A – Additional Sites:** Retests wafers based on a previously generated wafer description file. The previous wafer description filename is `LotID_retest.wdf`. The wafers are retested using (pattern number + `1`) of the original so that different sites are tested.
- **S – Runtime with Skip:** This test mode generates a random pattern for all wafers that have a wafer plan that matches the name found in the data pool string `KTXE_RP_random_wpf`. This mode must be used with absolute or relative wafer slot ID types.

KTXE_RP user library

Zone-based testing is accomplished with the use of calls to the user library `KTXE_RP` at various user access points (UAPs). The following routines are included in the `KTXE_RP` user library.

For more information about the commands in this user library, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`KTXE_RP_CleanUpWDF`	Renames the randomly generated WDF file to `lot_XXX_wdfname`.
`KTXE_RP_CreateRandomWDF`	Creates a wafer file based on the maximum frequency and number of wafers.
`KTXE_RP_CreateWPF`	Creates a new wafer plan file to use the appropriate probe pattern name.
`KTXE_RP_GetUsrArgs`	Checks the command-line arguments used to invoke the Keithley Test Execution Engine (KTXE) to determine the test mode.
`KTXE_RP_RemoveWPF`	Deletes wafer patterns that were generated at run time.

Required user access point files

A user access point (UAP) file, `adapt_zone.uap`, has been provided to illustrate the use of these library commands. This UAP file must be specified in the Keithley Test Program Manager (KTPM) to enable zone-based testing.

As an alternative to specifying this UAP file in every cassette plan file, this UAP file can be specified to affect all cassette plans executed. This is accomplished by setting the environment variable `$KI_KTXE_SYSTEM_AP` to point to the UAP file. Care should be taken in using the UAPs. Execution order is important. Refer to User access points for more information on UAPs and Environment variables (on page 6-155) for more information on environment variables.

Global data variables

There are three global data values that affect the behavior of zone based testing. These variables are normally defined in a global data file attached to the cassette plan file through Keithley Test Plan Manager (KTPM). These variables are as follows:

- `KTXE_RP_max_freq`: This variable defines the number of sites to be tested per wafer when using zone-based testing.
- `KTXE_RP_random_wpf`: This variable is only used by the Runtime Test with Skip method of zone-based testing. When using this test method, random wafer patterns will be generated for all wafers having a wafer pattern name matching the value of this variable.

- `KTXE_RP_test_mode`: This global data variable allows for all tests executed by the Keithley Test Execution Engine (KTXE) to be conducted using one of the zone-based testing modes. This has the same effect as using the four options for the command line argument `-u`. If you use `KTXE_RP_test_mode` to define the test mode, remove any `KTXE_RP_GetUserArgs` from the user access point (UAP) list. `KTXE_RP_GetUserArgs` will override this global data variable.

The following are the four valid values for `KTXE_RP_test_mode`.

Value	Command line equivalent	Test mode
0	N	Normal test
1	R	Run time
2	A	Additional sites
4	S	Run time with skip

Sample GDF files are provided illustrating the use of these global data values. For each Test Mode, a global data file with a name of the form `adapt_KTXE_RP_`*`n`*`.gdf` is included; where *`n`* is the test mode number.

Result-based testing

Result-based testing is a test method where the tester changes the sites and tests used in response to the test results received on previous sites. Result-based testing can be used to throw out bad wafers when a certain amount of failing data is received, minimizing the collection of bad data. It can also be used to reduce test time on good wafers by scaling back testing if all results are good. During data collection, results can be evaluated and additional tests run immediately when failing data is encountered. This can remove the need to re-probe the wafer.

Keithley Test Environment (KTE) uses six global variables in conjunction with the limits file to determine whether a site or wafer is good or bad. The limits file determines which parameters are monitored to determine failed sites. The global data variables determine how many tests can fail before a site fails, and how many sites can fail before the wafer fails. Complete descriptions of these global data variables are provided later in this section.

Result-based test modes

There are four test modes available for use with result based testing. These four modes operate as follows:

- **Alternate Sites on Current Wafer:** When testing on the current site fails, an additional alternate site is tested when the current site is completed. The alternate sites are pre-defined in the Wafer Description Utility (WDU) using alternate test patterns, and have a one-to-one relationship with the normal sites. Normal testing resumes on the next site.

- **More Sites on Current Wafer:** When testing on any site fails, additional predefined sites are tested. There is a one-to-many relationship between the normal sites and the alternate sites. Normal testing does not resume on the next wafer, but the additional sites will still be tested. The additional sites will continue for the remainder of the lot on all wafers with the same wafer plan, if no new wafer plan is encountered.
- **More Tests on Current Wafer:** When testing on a site fails, additional tests are used on the next site (current wafer only). On the next wafer, the original testing occurs (no additional tests will be performed). The additional testing occurs on the current wafer but not the next wafer because the function `KTXE_AT_wafer_begin` (at `UAP_WAFER_BEGIN`) sets the flag `KTXE_AT_alternate_site_test_mode` to `0`.
- **More Tests on Next Wafer:** When testing on a site fails, a new wafer plan will be executed on the next wafer, and all following wafer plans using the same name. This is done with a new wafer file.

Global data variables

There are six global data values that affect the behavior of result based testing. These variables are normally defined in a global data file attached to the cassette plan file through the Keithley Test Plan Manager (KTPM). These variables are as follows:

- `ktxe_at_ed_active`: This variable must be enabled (by setting to 1) to enable result based testing. When this global data item is enabled, tests using parameters marked as disabled in the Limits File Editor will not be performed. This variable is only used with the More Tests on Current Wafer test mode.
- `KTXE_AT_critical_param_failure_limit`: This variable defines how many critical parameters on a site must fail before the site itself fails. This variable is used with all result-based testing modes. To fail a site on any parameter failure, set the variable to 0. To permit two failures, set the variable to 2; the third failure will fail the site.
- `KTXE_AT_critical_site_failure_limit`: This variable defines how many sites on a wafer must fail before the wafer itself fails. If the wafer fails, testing on the wafer is aborted. This variable is used with all result based testing modes. To fail a wafer on any site failure, set the variable to 0. To permit two wafer failures, set the variable to 2; the third failure will fail the wafer. Note that critical site failures are only counted for the original sites.
- `KTXE_AT_limit_code`: This variable defines which specific limit is used for a critical parameter. The following values are used to determine which limit is used:

Value	Limit used
1	Valid
2	Spec
3	Control
4	Engineering

- `KTXE_AT_alternate_test_class`: This variable defines the class of extra tests to be enabled when a site fails. The class corresponds to the value of the Class field assigned in the Limits File Editor (LFE) to the extra tests that will be performed. The default value for this variable is `extra_tests`. This variable is only used with the More Tests on Current Wafer testing mode. For more information on LFE, refer to Limits File Editor (on page 6-78).
- `KTXE_AT_alternate_wafer_plan`: This variable defines the alternate wafer plan to be used when the next wafer will be tested with a new wafer plan. This variable is only used with the More Tests on Next Wafer testing mode.

There are three demo global data values that are used with `KTXE_AT_generate_val` to generate demo data. For more information, refer to the description of `KTXE_AT_generate_val` later in this section. These variables are as follows:

- `demo_type`: This variable is a number (`0` to `12`) which defines result failures based on site location on wafer.
- `demo_high_lim`: This variable defines the upper bound of passing results.
- `demo_low_lim`: This variable defines the lower bound of passing results.

Preparing result-based tests

Implementing result-based testing requires several additional steps beyond creating normal tests. Depending on the testing mode used, modifications will need to be made to the wafer definition file, wafer plan file, and limits file. Additional user access point (UAP) files and global data files (GDF) also need to be specified.

For information about using the Limits File Editor, refer to Limits File Editor (on page 6-78). For information on using the Wafer Description Utility, refer to The Wafer Description Utility (on page 6-15).

The following paragraphs describe the requirements for each of the modes of result-based testing.

General procedure for result-based testing

The following is a general procedure for setting up an existing cassette plan to use result-based testing. This procedure assumes that a cassette plan file has already been created using the tests to be included.

1. Start the Keithley Test Plan Manager (KTPM) and open the cassette plan file (CPF) file that will be adapted to use result-based testing.
2. Add the required user access point (UAP) file that corresponds to the test mode. The specific files required are listed in the description of the individual test modes.
3. If using the Alternate Sites Current Wafer or More Sites Current Wafer testing mode, modify the wafer definition file (WDF) file to include the alternate site pattern.
4. If using the More Tests Next Wafer testing mode, create a new wafer plan file to use as the alternate wafer plan.

5. Specify the required global data file (GDF) that corresponds to the test mode you want. The specific files required are listed in the description of the individual test modes. Set the global data variables as required by the testing mode you want.
6. Set up the limits file using the limits file editor (LFE). Define the critical parameters and test classes if required by the selected testing mode.
7. Save the cassette plan file.

When the cassette plan file is executed, the result-based testing mode will be enabled.

Alternate sites on current wafer

When using this test mode, the test results on each site are evaluated before moving to the next site. If the number of failed critical parameters on a site exceeds the value specified in `KTXE_AT_critical_param_failure_limit`, then the site fails. When a site is determined to have failed, the corresponding alternate site is tested.

The first step in implementing this test mode is to define the critical parameters using the limit file editor (LFE). This is done by specifying a value in the CR field for each parameter to be evaluated for result based testing. Valid values for the CR field are `N` and `1` through `9`. For the purposes of result-based testing, all values other than `N` are treated as critical parameters.

After specifying the critical parameters, alternate test sites need to be specified. Alternate test sites are created by creating a new pattern and adding an `_ALT` to the pattern name. The alternate pattern must have the same number of sites as the original pattern.

After specifying the alternate sites, the appropriate global data values need to be specified. The global data item `KTXE_AT_critical_param_failure_limit` must be set to the number of critical tests to fail on a site before the site itself is considered to have failed.

After all the component files have been set up, the test is set up as normal with the Keithley Test Plan Manager (KTPM). The only exception is that the appropriate global data files (GDF) and user access point (UAP) files must be added to the test plan. For this test mode, the GDF and UAP files to be added are `adapt_alternate_site.gdf` and `adapt_alternate_site.uap`.

More sites on current wafer

When using this test mode, the test results on each site are evaluated before moving to the next wafer. If the number of failed critical parameters on a site exceeds the value specified in `KTXE_AT_critical_param_failure_limit`, then the site fails. When any site fails, all of the alternate sites are tested.

The first step in implementing this test mode is to define the critical parameters using the limits file editor (LFE). This is done by specifying a value in the CR field for each parameter to be evaluated for result-based testing. Valid values for the CR field are `N` and `1` through `9`. For the purposes of result-based testing, all values other than `N` are treated as critical limits.

After specifying the critical parameters, alternate test sites need to be specified. Alternate test sites are created by creating a new pattern and adding an `_ALT` to the pattern name. The alternate pattern can have more sites than the normal pattern.

After specifying the alternate sites, the appropriate global data values need to be specified. The global data item `KTXE_AT_critical_param_failure_limit` must be set to the number of critical tests to fail on a site before the site itself is considered to have failed.

After all the component files have been set up, the test is set up as normal with the Keithley Test Plan Manager (KTPM). The only exception is that the appropriate global data files (GDF) and user access point (UAP) files must be added to the test plan. For this test mode, the GDF and UAP files to be added are `adapt_more_sites_cur_wafer.gdf` and `adapt_more_sites_cur_wafer.uap`.

More tests on current wafer

When using this test mode, the test results on each site are evaluated before moving to the next site. If the number of failed critical parameters on a site exceeds the value specified in `KTXE_AT_critical_param_failure_limit`, then the site fails. When a site is determined to have failed, additional tests are immediately run on the site. The additional tests will continue to be run on all sites on the present wafer, and normal testing resumes on the next wafer.

The first step in implementing this test mode is to define the critical parameters using the limits file editor (LFE). This is done by specifying a value in the CR field for each parameter to be evaluated for result-based testing. Valid values for the CR field are `N` and `1` through `9`. For the purposes of result-based testing, all values other than `N` are treated as critical limits.

In addition to specifying the critical parameters, the additional tests must also be specified. These are specified by setting the eClass of the parameter to the value defined in the global data item `KTXE_AT_alternate_test_class`. The eClass is set to `extra_tests` by default. Tests associated with this class will not be performed until a site fails.

After specifying the additional tests, the appropriate global data values need to be specified. The global data item `ktxe_at_ed_active` must be enabled. The global data item `KTXE_AT_critical_param_failure_limit` must be set to the number of critical tests to fail on a site before the site itself is considered to have failed. The global data item `KTXE_AT_alternate_test_class` must be set to the class used in the Class field in the LFE.

After all the component files have been set up, the test is set up as normal with the Keithley Test Plan Manager (KTPM). The only exception is that the appropriate global data files (GDF) and user access point (UAP) files must be added to the test plan. For this test mode, the GDF and UAP files to be added are `adapt_more_tests_cur_wafer.gdf` and `adapt_more_tests_cur_wafer.uap`.

More tests on next wafer

When using this test mode, the test results on each site are evaluated before moving to the next wafer. If the number of failed critical parameters on a site exceeds the value specified in `KTXE_AT_critical_param_failure_limit`, then the site fails. When a site is determined to have failed, testing on all future wafers in the lot with the same wafer plan name will use an alternate wafer plan.

The first step in implementing this test mode is to define the critical parameters using the limits file editor (LFE). This is done by specifying a value in the CR field for each parameter to be evaluated for result-based testing. Valid values for the CR field are `N` and `1` through `9`. For the purposes of result-based testing, all values other than `N` are treated as critical limits.

After specifying the critical parameters, the appropriate global data values need to be specified. The global data item `KTXE_AT_critical_param_failure_limit` must be set to the number of critical parameters to fail on a site before the site itself is considered to have failed. The global data item `KTXE_AT_alternate_wafer_plan` must be set to the name of the alternate wafer plan to be used if a site fails.

After all the component files have been set up, the test is set up as normal with the Keithley Test Plan Manager (KTPM). The only exception is that the appropriate global data files (GDF) and user access point (UAP) files must be added to the test plan. For this test mode, the GDF and UAP files to be

KTXE_AT user library

Result-based testing is accomplished with the use of calls to the user library KTXE_AT at various user access points (UAPs). The following routines are included in the KTXE_AT user library.

For more information about the commands in this user library, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`KTXE_AT_alternate_site_site_end()`	When a site has finished testing, determines if an alternate site should be tested.
`KTXE_AT_alternate_site_test_end()`	Determines if any test results failed.
`KTXE_AT_AlterWWP()`	Alters working wafer plans when Keithley test modules fail (used internally).
`KTXE_AT_CheckResWithLimits()`	Checks the value of test results with specified limits.
`KTXE_AT_cleanup_site()`	Resets counts and cleans up data structures (used internally).
`KTXE_AT_debug_print()`	Prints internal flags and counters.
`KTXE_AT_demo_data_func()`	Generates demo data (used internally).
`KTXE_AT_enable_kdf()`	Enables `.kdf` logging.
`KTXE_AT_FindAltSite()`	Finds alternate coordinates for present site (used internally).
`KTXE_AT_LogResultList()`	Logs saved result lists to a `.kdf` file (used internally).
`KTXE_AT_more_sites_cur_wafer_site_end()`	Determines if more sites should be tested on the present wafer.
`KTXE_AT_more_tests_curr_wafer_site_end()`	Determines if more test should be done on the present wafer.
`KTXE_AT_more_tests_curr_wafer_wafer_begin()`	Adds results to list of results disabled during initial execution.
`KTXE_AT_more_tests_next_wafer_site_end()`	Changes wafer plan for next like-named wafer plan.
`KTXE_AT_wafer_begin()`	Initializes the KTXE_AT user library module.

Installation of adaptive testing user libraries

To install the adaptive test user library for zone-based testing, enter the following at the command line:

```
> cd $KIHOME/src/KTXE_RP
> kult_copy_lib -lKTXE_RP
```

To install the adaptive test user library for result-based testing, enter the following at the command line:

```
> cd $KIHOME/src/KTXE_AT
> kult_copy_lib -lKTXE_AT
```

Installation of demonstration files

Keithley has provided a collection of demonstration files for zone and result-based testing. To install these files, execute the script `KTXE_RP_demo_install` for zone-based files and `KTXE_AT_demo_install` for result-based files. These scripts will copy the files for these demos to the current projects directories.

To install files for zone-based demo, enter the following at the command line:

```
> cd $KIHOME/src/KTXE_RP
> KTXE_RP_demo_install
```

To install files for result-based demo, enter the following at the command line:

```
> cd $KIHOME/src/KTXE_AT
> KTXE_AT_demo_install
```

The installed demo files for adaptive testing start with `adapt_`.

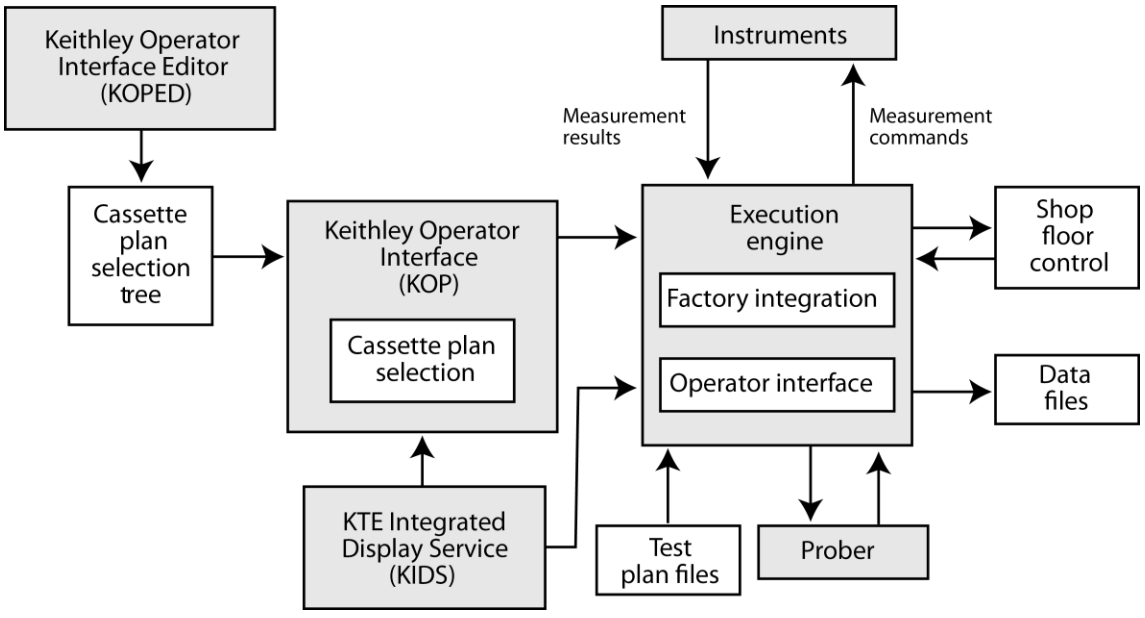
Test execution

You can execute a cassette plan in one the following ways:

- Using Keithley Integrated Display Service (KIDS)
- Using Keithley Operation Interface Editor (KOPED) and Keithley Operator Interface (KOP)
- Directly accessing the Keithley Test Execution Engine (KTXE)

The following figure shows a diagram of the test execution process.

Figure 76: Test plan execution flow diagram



This section describes the tools required to execute a completed test plan. In this section, the following will be discussed:

- **Keithley Operator Interface Editor (KOPED)**: How to create a series of test processes that can be activated from the Keithley Operator Interface (KOP).
- **Keithley Test Execution Engine (KTXE)**: How to use KTXE to execute the test plans created using the Keithley Test Plan Manager (KTPM).
- **KTE Integrated Display Service (KIDS)**: How to use KIDS for recipe selection, execution, and display status.

Keithley Operator Interface Editor (KOPED)

Keithley Operator Interface Editor (KOPED) is a graphical user interface used by the test engineer to create a test initialization file. The information entered into the initialization file determines the operator choices available when the Keithley Operator Interface (KOP) is run. There are three levels in the selection tree available to be set up: Level 1, Level 2, and Level 3. These can be set to any string you want.

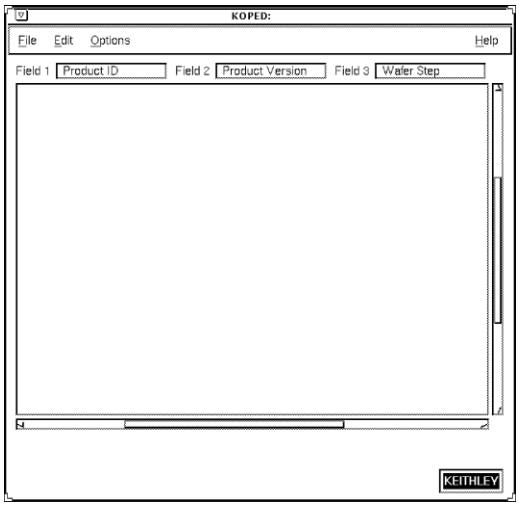
KOPED main window

The Keithley Operator Interface Editor (KOPED) main window, shown in the following figure, contains the following areas:

- **Main menu bar**: Contains the File, Edit, Options, and Help menus.
- **Data Entry Level 1**: Used to enter the label for column one in the Keithley Operator Interface (KOP).
- **Data Entry Level 2**: Used to enter the label for column two in KOP.
- **Data Entry Level 3**: Used to enter the label for column three in KOP.
- **Window work area**: Shows a flow tree for all of the possible test routines that can be accessed through KOP.
- **Engineering data entry Level**: Used to enter the name that appears in the highlighted box in the tree in KOPED.
- **Plan or Program (with path)**: Used to enter the cassette plan filename or executable program name and directory path.

The Engineering and Operator Levels do not appear until a Level 1 is created or an existing `.ini` file is loaded. The Plan field will appear when an entry in Level 3 is created.

Figure 77: KOPED main window



KOPED File menu

The Keithley Operator Interface Editor (KOPED) File menu contains the following selections:

- **New**: Clears the KOPED main window to create a new initialization file.
- **Open**: Opens an existing initialization file.
- **Save As**: Saves the present initialization file using a new filename.
- **Save**: Saves the present initialization file using the present filename.
- **Exit**: Exits KOPED.

KOPED Edit menu

The Keithley Operator Interface Editor (KOPED) Edit menu contains the following selections:

- **Window label**: Lets you enter the window label for the operator interface (when KOP is run).
- **Delimiter**: Lets you enter the delimiter of the initialization file.
- **Add Level 1**: Lets you add a new top-level field to the main window work area.

KOPED Options menu

The Keithley Operator Interface Editor (KOPED) Options menu lets you enable or disable Test Plan Validation. If enabled, the operator will be able to use the Verify button on the Keithley Operator Interface (KOP) screen to validate the information in a cassette plan before execution. If disabled, the Verify button on the KOP screen will be disabled, preventing the operator from cassette plan validation.

KOPED Help menu

The Keithley Operator Interface Editor (KOPED) Help menu provides online help and KOPED version information. The Documentation option allows access to online Help. The About option displays the release version.

Pop-up menus

With the exception of the Level 1 pop-up menu, the pop-up menus are accessed by placing the cursor in the boxes, and pressing and holding the right mouse button. Selections are made by pulling the cursor down to the selection and releasing the mouse button.

The New Level 1 pop-up menu is accessed by pressing and holding the right mouse button anywhere in the window area of the Keithley Operator Interface Editor (KOPED). The pop-up menus include:

- **New Level 1 pop-up menu**: Lets you Add or Paste Level 1 data.
- **Edit Level 1 pop-up menu**: Lets you Cut or Copy data, and Add or Paste Level 2 data.
- **Edit Level 2 pop-up menu**: Lets you Cut or Copy data, and Add or Paste Level 3 data.
- **Edit Level 3 pop-up menu**: Lets you Cut or Copy data.

Keithley Test Execution Engine (KTXE)

KTXE is a graphical user interface used by the test engineer to execute a cassette plan file (`.cpf`) that can be created using the Keithley Test Program Manager (KTPM).

It is possible to define an alternate localization file. Refer to Keithley Test Environment (KTE) Keithley UI (KUI) localization in Keithley User Interface Library.

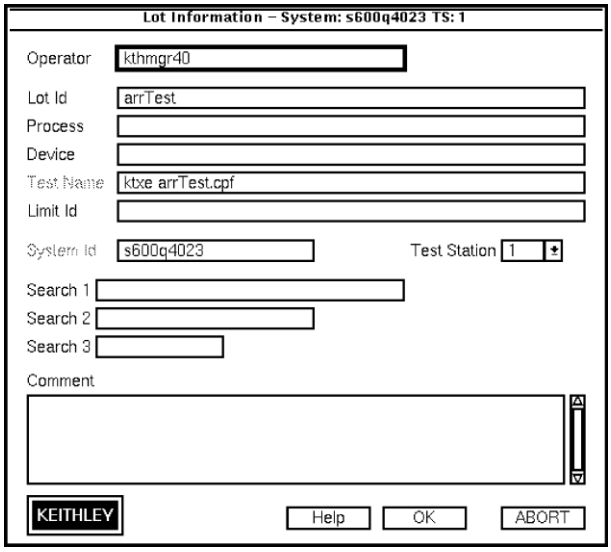
The KTXE interface can be viewed in any language. Refer to KTE KUI localization (on page 6-199).

KTXE main window

The Keithley Test Execution Engine (KTXE) main window shown in the following figure contains the following areas:

- **Operator data field:** This field is used to enter the ID of the operator running the test.
- **Lot Id data field:** This field is used to identify the test path and filename. This data is entered when the cassette plan file is specified when KTXE is started.
- **Process data field:** This field is used by the operator to enter information about the test process.
- **Device data field:** This field is used by the operator to enter any necessary information about the device.
- **Test Name data field:** This field is used to identify the test name. This data is entered when the cassette plan file is specified when KTXE is started.
- **Limit Id data field:** This field is used by the operator to enter the limit file that will be used during testing.
- **System Id data field:** This field is used to identify the system that is running the test.
- **Test Station field**: This field is used to select the test station that the test will be run on.
- **Search key data fields:** The search keys are used to further identify the test process when test results are being searched for in the Keithley Summary Utility (KSU).
- **Comments field:** This field is used to enter any comments about the test being run.
- **Control buttons:** Used to start the test process, abort the test process, and to receive online help for KTXE.

Figure 78: Lot dialog window

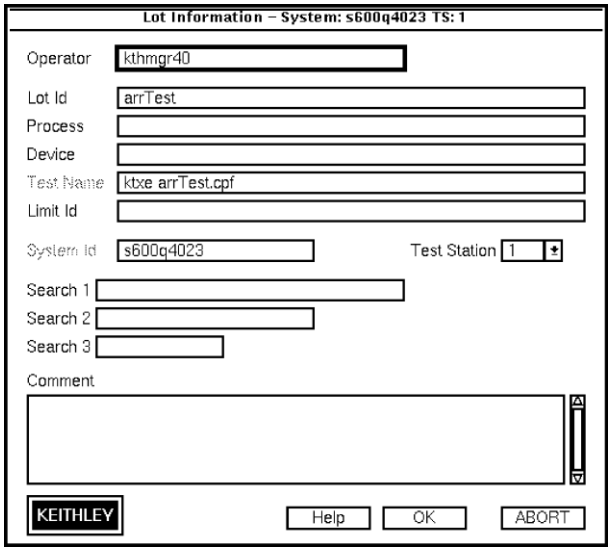


Test execution from KTXE

It is also possible to execute your cassette plan by directly accessing the Keithley Test Execution Engine (KTXE).

1. Start KTXE from the command line by entering the following: `ktxe -cpf fname.cpf` where `fname.cpf` is the name of the cassette plan.
2. Verify all the test execution data that appears in the Lot Dialog window, as shown in the following figure.

Figure 79: Lot dialog window



3. Click **OK** to begin the testing process.

Command-line options for KTXE

The following is a listing of all the command-line options that can be entered when starting the Keithley Test Execution Engine (KTXE).

Switch	Type	Description
-c	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot information comment field; this switch can be used to add comment text to the lot header.
-cass	n (numeric)	Select cassette n: 1 through 4; this switch is used to select a specific cassette for a prober with multiple cassettes.
-cpf	fname (valid filename and path)	Cassette plan filename; this switch is required if the -krf switch is not used, and it specifies the name of the cassette plan to execute.
-d	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot information device field; this switch can be used to specify the device field of the lot header.
-doc	"options" (must be enclosed in quotation marks)	Documentation tool information; this switch is used to disable portions of the execution to allow test documentation information to be collected. These switches can also be used to customize KTXE execution. ■ NO_LOT: Disable default lot reporting .kdf ■ NO_KUI: Disable the user interface ■ NO_PROBER: Disable Prober activity ■ NO_KTM: Disable Keithley test module (KTM) execution ■ NO_UAP: Disable user access point (UAP) execution ■ DOC_ON: Enable Test Plan Document data generation; this will cause KTXE to create a data file containing execution information. This data file is named casPlanName.cpf.kxd and is placed in the directory specified by the $KITMP environment variable. ■ DOC_ONLY: Configure for Test Plan Document generation only Several of the above -doc options can be used together. For example, to disable the default .kdf logging and default prober activity, use the following command-line switch for KTXE: -doc "NO_LOT,NO_PROBER"

Switch	Type	Description
-e	n [fname] (numeric) (valid filename and path)	Error reporting mode n: 0 through 3 ■ 0: None ■ 1: Display messages ■ 2: Log messages ■ 3: Display and log message
-ev	n [fname] (numeric) (valid filename and path)	Event reporting mode n: 0 through 3 ■ 0: None ■ 1: Display messages ■ 2: Log messages ■ 3: Display and log message The -e and -ev switches enable the error and event logging. The error and event reporting capability of KTXE can be used to determine possible problems with macros and test plans.
-h		Display command line options (shown here)
-i	id (valid filename, no paths or extensions)	Lot information lot id field; the lot id determines the name of the .kdf lot file; the actual .kdf lot filename is $KI_KTXE_KDF/lot_id.kdf
-k	n text (numeric) (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot search key n: 1 to 3 field; this switch allows the search key fields in the .kdf lot file to be filled with the specified text; use a separate -k for each field
-krf	fname (valid filename and path)	Uses the specified recipe name; see Recipe Manager (on page 7-1) for more information
-l	id (valid filename, no paths or extensions)	Lot information limit id field; this switch will specify the limits file to use for processing test data; the limits file specified by this switch will override any limits files specified in wafer plan files
-o	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot information operator field; this switch allows the operator field in the .kdf lot file to be filled with the specified text
-p	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot information process field; this switch allows the process field in the .kdf lot file to be filled with the specified text
-r	"options" (must be enclosed in quotation marks)	Lot summary report options; this switch allows the sum_report_options data pool entry to be filled with the text string specified; calling the KTXEAddIn:KTXESummaryReport() function at UAP_LOT_END allows the LotSummary program to be executed with the specified options
-s	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	Lot information system field; this switch allows the system field in the .kdf lot file to be filled with the specified text

Switch	Type	Description
-u	text (command-line arguments are concatenated to the switch argument until the next switch occurs)	User argument; this switch can be used for additional user command-line arguments
-w	fname (valid filename and path)	Wafer description filename; this switch will specify the wafer description file to use for processing test data; the wafer description file specified by this switch will override any wafer description file specified in the cassette plan or wafer plan file
-x	n (numeric)	Debug flag n: 0 to 32767; this switch can be used for user command-line arguments

The execution process

Once test execution has begun, the execution engine begins working its way through the cassette plan you specified. The Keithley Test Execution Engine (KTXE) uses the engine you selected for your cassette plan as the format for how KTXE will proceed when accessing and executing the data files specified in the cassette plan file. The following figure shows a flow diagram of how the execution engine proceeds through all of the test data.

Figure 80: Test execution engine

The execution engine begins at the cassette level, gathering cassette information and collecting lot and prober information. The engine then moves on to the wafer level. Here, the engine loads all the data specified in the wafer plan. When all the wafer information is loaded, the engine moves on to the Keithley test module (KTM) level where the test execution process begins. The engine finds each specified site and subsite, and then performs the specified test at each selected position. The execution engine continues this process until either it has exhausted all the information specified within the cassette plan or logged test data has signaled an abort.

The execution engine will order probing and testing based on the following:

- If the Optimize Site Processing button in the Wafer Definition Utility (WDU) is selected, the execution engine will process sites based on Y and X coordinate positioning. If the Optimize Site Processing button is not selected, the site order will be as specified in the Probe Pattern Editor in the WDU (first occurrence, first probed). In both cases, each site will be probed only one time, as the execution engine merges all tests across probe patterns.
- Tests will occur in the order they are specified in the Wafer Plan Editor (Keithley Test Plan Manager, KTPM). Subsites will be probed in the order they are specified in the Site Editor (WDU).

The following is an in-depth description of what happens at each user access point (UAP) during test execution. Each UAP is listed, followed by a description of the actions that UAP causes to occur.

KTXE INITIATED

- Initialize engine variables.
- Set data pool value "ManualProberType" to `FALSE`; use global data to set otherwise.
- Process the command-line arguments.
- Initialize error logging.
- Determine Documentation Tool command-line options and configure the execution engine appropriately. First store the default values into the Data pool. The `GetDocumentationConfig` routine will adjust the values if necessary.
- Load the cassette plan.
- Load global data files only if there are files to load.
- Refresh the "ManualProberType" variable from Global Data Load.

UAP_PROG_ARGS

- Get LOT ID. Check command line first, then the cassette plan.
- Check for Suspended Lot information.

UAP_CASSETTE_LOAD

- Check for empty slot list. Exit if empty.
- Collect any additional lot related information at this user access point (UAP) before displaying lot dialog screen to operator.

UAP_LOT_INFO

- Display Lot dialog screen and gather information from the operator.
- Display status dialog on the screen.
- Starting Tester Control.
- Add the LOT ID to the data pool.
- The product filename should be set at this user access point (UAP) if used.

UAP_PROBER_INIT

- If the "`KI_PRB_AUDIT_LOG`" environment variable is set, then:

  ```
  EnableTransactionLogging();  /* prober transactions */
  ```

- Load the product file. If it is non-null:

  ```
  if (product_file[0] != "\0")
  ```

- `KTXELoadProductFile(product_file);`

> **NOTE**
>
> Check options after product file load because product file could enable or disable options.

- Check Prober Options.
- Prompt operator to load Cassette onto Prober.
- Determine the slot mode: all (ALL), index (nn), or relative (Rnn).
- Check only first slot.

  ```
  wafer->boat = 1; /* assume cassette 1 only */
  ```

  ```
  FindWafersToProbe -
  ```

  Determine which wafers are to be probed and set the prober's slot status accordingly.

- Get a Cassette Mapping from the prober regardless of the slot mode.
- Error if "Cassette not loaded," if there are "unmapped wafers," or if "No Unprobed wafers found," try again.
- Determine the slot mode: all (ALL), index (nn), relative (Rnn)

- Check only first slot in the slot list.
- If "ALL" mode "where All Mode means test ALL WAFERS"

  Build a slot list using all available wafers in cassette.

- If "Relative" Mode

  find the nth slot with a wafer in cassette.

NOTE

If no more wafers are found unprobed, no errors are generated. PrLoad Cassette complete will handle the completion. Any non-relative slots will be skipped.

- If "indexed slot" mode:

  set slot status IF a wafer is really there.

- Send new cassette slot status to prober.

UAP_WAFER_MISMATCH

- END FindWafersToProbe
- Load the Wafer Description File so the prober can be initialized.

Hierarchy of loading:

1 - Command-line arg.

2 - Cassette Plan.

3 - Wafer Description File of *FIRST* Wafer Plan.

UAP_ACCESS_WDF_INFO

- This UAP allows access to the .wdf file before the call to PrInit.
- Initialize the Prober (PrInit).

UAP_POST_PROBER_INIT

- Any extra changes to the LOT header should be made at this user access point (UAP).

UAP_WRITE_LOT_INFO

- Write LOT structure into `.kdf` file.
- Register exit handler for `.kdf` (EndWafer).

UAP_POST_LOT_INFO

- This UAP may be used to add tag data to the lot file before the first wafer is tested.
- Do we skip the first wafer?

  Reload from data pool in case a UAP modified the value.

  This data pool value can be set for probers that need the first wafer manually loaded.
- If not skip_first_wafer_load

  Load Wafer

UAP_WAFERLOAD_STATUS

- if (TRUE == ManualProberType)
- Prompt operator to "Load/Unload Wafer from chuck".

UAP_PROFILE_WAFER

- This UAP may be used to provide time for the hot chuck to warm up the wafer before profiling.
- Profile first WAFER.
- Align first WAFER.

UAP_POST_INITIAL_WAFER_LOAD

- Perform AutoZ after the first wafer load.

NOTE

AutoZ is a function of the SofTouch optional licensed feature for the Keithley Test Environment (KTE).

- Loop through slots defined in the cassette plan.

  Wafer Loop

UAP_ALIGN_ERROR

- Error recovery after wafer alignment.

UAP_WAFER_PREPARE

- Get wafer plan name from the slot list entry.
- Keep count of wafers defined in cassette wafers_tested++.
- Determine if the previous wafer plan is the same as the current wafer plan.
- If the previous wafer had the same wpf, do not reload the working wafer plan (WWP), test enable all tests in WWP since there may have been a previous abort.

- if (previous_wafer_has_same_wpf == FALSE)

  Load the wafer plan for the wafer.

  Load the probe card file.

  Load the Limits filename.

  Update the `limit_list` data pool pointer.

  Load Wafer Description File.

  Prepare a complete wafer plan for execution.

  Load all the Keithley test modules (KTMs) in wafer patterns.

  Keep track using the data pool of the KTM list.
- Place the address of the wwp into the data pool.

  ```
  dpAddPointer( "wwp_list", LONG_P, wwp_list );
  ```
- Read wafer ID from prober.

UAP_VALIDATE_OCR

- Start of the wafer tests (`start_wafer_test`:).
- Reset the `KI_ktxe_retest_wafer` data pool flag.
- Write wafer header to the LOT file.

UAP_WAFER_BEGIN

- Execute a wafer plan.
- Add `result_list`, `failed_result_list` and abort level to data pool for use at `UAP_HANDLE_ABORT`.
- Get site and subsite pointers from the data pool.
- Get a working copy of `wwp_list` and place into data pool as "`current_wwp_list`".
- LOOP: (Execute only tests that are enabled in wafer plan)
- Reset "`KI_ktxe_redo_macro`" data pool flag.
- Reset "`ktxe_disable_kdf`" data pool flag to initial value.
- Move probe chuck to the test site if x or y values changed.

UAP_SITE_CHANGE

- Write site data to LOT file.
- Move probe chuck to the test subsite if changed.
- Raise the chuck to make contact with pins.

UAP_SUBSITE_CHANGE

- Save copy of this subsite as the prev or last subsite.

UAP_TEST_BEGIN

- Execute the Keithley test module (KTM).
- Update the `result_list` DataPool value with new results.

UAP_TEST_END

- Write results to LOT file.

UAP_TEST_DATA_LOG

- If (`KI_ktxe_skip_limits_check = = 0`), check limits file results for abort conditions.
- Disable tests based on abort flags. If subsite or site aborts, disable tests in the working wafer plan (WWP).

UAP_HANDLE_ABORT

- if WAFERABORT we will leave execution loop.
- if LOTABORT we will leave execution loop.
- if (ktxe_abort_logging)

  log Abort Reason
- If "KI_ktxe_redo_macro" data pool flag is not set then

UAP_SUBSITE_END

At end of subsite processing, this access point is called.

UAP_SITE_END

- increment to next test.
- Update "current_wwp_list" data pool value with new entry.
- END LOOP: (Execute only tests that are enabled in wafer plan)
- Do a Prober PrRelReturn to clean up for the next wafer.
- Call EndSite to close up KDF file.

  End of execute a wafer plan.

UAP_WAFER_END

- Handle wafer and lot abort as needed.
- Write end of wafer to LOT file.
- Check if we want to redo this wafer using the `KI_ktxe_retest_wafer` data pool flag.

  if so

  go to `start_wafer_test`

  else

  load next wafer

UAP_PROFILE_WAFER

- This user access point (UAP) may be used to provide time for the hot chuck to warm up the wafer before profiling.
- Profile next WAFER.
- Align next WAFER.

UAP_WAFERLOAD_STATUS

- If (TRUE == ManualProberType.
- Prompt operator to "Load/Unload Wafer from chuck."

UAP_ALIGN_ERROR

- If cassette is complete...

  Clean up the working wafer plan (WWP) list

  else

  Advance to next slot in slot list.
- If no more slots, or next slot uses different Wafer Plan:

  Clean up the WWP list

  Set flag saying different WPF for next slot

  else

  Next wafer exists and uses same wafer plan so Enable Tests in WWP.

  End of Wafer Loop.
- Write end of lot to LOT file.

UAP_LOT_END

- Release the tester using the `tstdsl()` command
- Set data pool flag for no error...normal exit mode...

UAP_ENGINE_EXIT

- Pause the status dialog so the operator can inspect the display.
- Clean up internal memory structures.
- Release user interface support.
- Set flag so the AbortExitHandler routine will skip execution

  NormalExit = TRUE ;

  exit(KI_OK);

NOTE

The following UAPs (UAP_ABORT_EXIT_HDLR, UAP_PRB_ERR_HDLR, and UAP_STATUS_CHANGE) are executed at the time the event occurs.

UAP_ABORT_EXIT_HDLR

- At UAP_ABORT_EXIT_HDLR, KUI functions may not be used to display information. An external program may be used to check Keithley Test Execution Engine (KTXE) exit status and prompt with a GUI if necessary.

```
void KTXEAbortExitHdlrStub()
{
  We only want to do this if we abort...
}
```

NOTE

Here is the list of the `atexit()` functions that we use and their order of placement onto the stack. The execution order is the reverse.

Install order:

- AbortExitHdlr
- tstdsl
- Endlot

Execution order:

- Endlot
- tstdsl
- AbortExitHdlr routines

Because the abort exit handler routine is the last to execute, if the user wishes to do prober or LPTLib calls, they will have to perform a tstsel call first and then a tstdsl call to release.

Also, note that we are executing this abort exit routine because the KTXE process is being terminated. Certain system items may not be stable at this time. For this reason, KUI calls are illegal at this user access point (UAP).

UAP_PRB_ERR_HDLR

- If a prober error occurs, this user access point (UAP) is used to call user created recovery code.

UAP_STATUS_CHANGE

- This user access point (UAP) is called when the operator presses the Pause or Continue buttons on the Status Dialog Window, causing an execution state change. This UAP can be used to notify a shop-floor control system that the tester has been paused.

User library files required for KTXE execution

During Keithley Test Execution Engine (KTXE) execution, not all user library files are required to be on a tester. If a file server is used to store and develop user libraries, it may be convenient to only copy the files needed for executing tests.

The following user library files are required by KTXE in the `KI_KULT_PATH` directory at execution time:

- `libaaaa.so`
- `libkittaaaa.so`
- `aaaa_proto.h`
- `aaaa_paramset.psf` (if parameter sets are used from this library)

Where "`aaaa`" is the name of the user library.

If the `KI_KULT_PATH` is not mapped to the default location, the following Keithley provided files must be in the `KI_KULT_PATH` directory.

Prober user library:

- `libprbxxxx.so`

Where "`xxxx`" is the prober driver type in use. Example: "`xxxx`" equals "`EG40`" for the Electroglas 4090 prober.

Keithley provided user library:

- `libuuuu.so`
- `libkittuuuu.so`
- `uuuu_proto.h`

Where "`uuuu`" is any Keithley user access point (UAP) user library. Example: "`uuuu`" equals "`KTXEAddIn`" for `libKTXEAddIn.so`, `libkittKTXEAddIn.so`, and `KTXEAddIn_proto.h`.

The KTXE ErrorHandler function

The Keithley Test Execution Engine ErrorHandler function will output additional information to you. A data pool item, `ktxe_error_gui`, can be set by you and will have the following effect:

- If `ktxe_error_gui` is set to `1`: The operator will be prompted with a choice of continuing with the error or aborting the test program.
- If `ktxe_error_gui` is set to `2`: The operator will be notified that the test program will abort. The operator can only select OK.
- If `ktxe_error_gui` is set to any other value or not defined at all, the error will be logged to the KTXE error log.

In addition, if KUI (`ktxe_disable_kui` controls, if the GUI is used) is disabled, output to the operator will be disabled.

In general, this functionality will be available in conjunction with selected errors that the operator may be able to correct or control.

Error and event logging

With the Keithley Test Execution Engine (KTXE), it is possible to create a log containing a report of the events and errors that occur during test execution. There are two ways to set the path for the event logs:

- Specify the log path at the command line when starting KTXE.
- Set the environment variables so they contain the log path.

NOTE

If the "`-e`" (error) and "`-ev`" (event) command line options are specified, the filename specified with these options override the filename set by the environment variables.

To set the log path at the command line when KTXE is started, enter:

```
ktxe -cpf cassettePlan -e 2 /mydir/file.log -ev 2 /mydir/file.log
```

This enters the events (`-ev`) and errors (`-e`) into the log file `file.log`.

To accomplish the same event and error logging with the environment variables, enter:

prompt> `setenv KI_KTXE_EVENT_LOG /mydir/file.log`

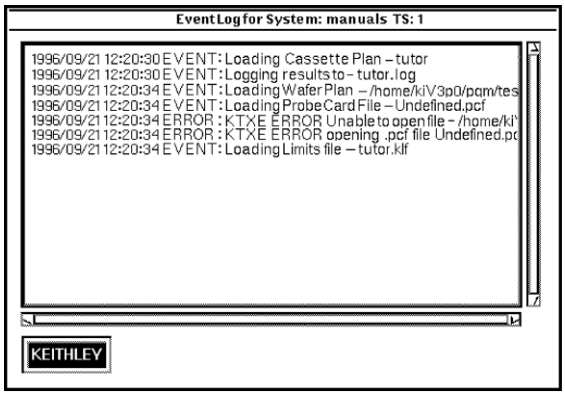
prompt> `setenv KI_PRB_AUDIT_LOG /mydir/file.log`

prompt> `setenv KI_KTXE_ERROR_LOG /mydir/file.log`

prompt> `setenv KI_LPT_DEBUG_LOG /mydir/file.log`

An example of an event log is shown in the following figure.

Figure 81: Event log window



Lot suspend and resume

When the Abort button is pressed on the Status Dialog window, the operator can abort immediately, suspend, or cancel. If suspend is selected, the Keithley Test Execution Engine (KTXE) will stop execution at completion of the current macro. Data files are created to keep track of current execution status, so if the engine is started with the same cassette plan again, the operator is asked if they would like to resume the suspended lot.

⚠ WARNING

The `.kdf` file will contain multiple entries for the wafer/site that execution was suspended on.

Example:
Assume that each wafer has 5 sites, 4 subsites per site, one macro per subsite.

Assume execution was suspended on wafer 2, site 4, subsite 2. Then the lot execution was resumed to completion.

The Keithley data file (KDF) file will contain data for wafer 1, wafer 2 up to site 4, and parameters including data for subsite 2. Then, there will be another entry for wafer 2, site 4 and parameters for subsite 3 and 4, then site 5, and the rest of the lot data.

Rules:
The execution status files are named `casPlanName_QMO.sav` and `casPlanName_QMO.wwp`, located in `$KI_KTXE_CPF`. This means that you cannot suspend LOT1 using `casPlanName` and then run and suspend LOT2 using `casPlanName`. The data for LOT1 suspension will be lost. You must resume a test before suspending another test using the same cassette plan.

Multicassette Multilot Testing Utility

The multicassette multilot Keithley Test Execution Engine (KTXE) launcher (`multi_cassette`) utility will collect the standard lot dialog information for one to four cassettes to be tested.

For each lot the wafer ids for each wafer may be entered. After all data is available, the cassettes will be tested without further information from the operator. Cassette plans that are launched by the `multi_cassette` program require the user access point (UAP) module `KTXEExtSetWaferId` from the KTXEAddIn library to be placed at the `UAP_WRITE_LOT_INFO`.

Also, to disable the standard operator dialog boxes used during testing, the following items should be placed in a global data file (`.gdf`) included in the cassette plan:

```
ktxe_disable_load_cassette_msg,INT,1
display_lotdlg,INT_P,0
ktxe_disable_plan_complete_msg,INT,1
```

Sources for the `multi_cassette` program are included in the `$KIHOME/src` directory if customization is required.

To rebuild the `multi_cassette` program enter:

```
> compile_ktxe.sh multi_cassette.c
```

Wafer id usage in KTXE

The Keithley Test Execution Engine (KTXE) will query the prober, lookup in the cassette plan, or generate a wafer id during execution.

If the prober has a wafer id reader, the id is returned to the `prober_wafer_id` data pool value if non-null. Or, if the cassette plan contains, or the operator entered, a wafer id, the wafer id is returned to the `prober_wafer_id` data pool value. Or, a wafer id is generated in the form "`Wafer_nn`", where "`nn`" is the cassette slot number.

At `UAP_VALIDATE_OCR` the data pool value `prober_wafer_id` can be checked or altered. See Data pool (on page 6-228) for more data pool usage information.

Abort flags

The Keithley Test Execution Engine (KTXE) has the ability to check the results generated from the execution of a test macro against the specified limits file. If a certain result is out of the acceptable limits range, KTXE takes the appropriate abort action as specified in the limits file. The following is a more detailed description of how limits checking and abort conditions are handled by KTXE.

Limits checking

After the Keithley Test Execution Engine (KTXE) executes a test macro at a subsite, it checks all the results against the limits specified in the limits file. For each result, the acceptable range is determined using the specified abort limit for that result. A linked list of all the failed results is created. If a result is outside the acceptable limit, it is added to the failed results linked list. Also, the highest level of the abort action is determined out of all the failed results, and that abort action is put into the data pool. At the user access point `UAP_HANDLE_ABORT`, which is encountered after the execution of each of the test macros, you have access to the failed results list and the abort action flag.

At the user access point `UAP_HANDLE_ABORT`, you can extract useful information regarding the failed results and determine the highest level of abort (for example, LOT, WAFER, SITE or SUBSITE) caused by the failed results. There are five different abort levels. The following is the list starting from highest to lowest.

NOTE

Although there are five abort levels, only the first four listed have failed results. Therefore, you may have action taken based on the abort level using `UAP_HANDLE_ABORT` for the first four levels (not NOABORT).

- **LOTABORT:** Testing on the lot is aborted due to the failed result.
- **WAFERABORT:** Testing on the current wafer is aborted due to the failed result.
- **SITEABORT:** Testing on the current site is aborted due to the failed result.
- **SUBSITEABORT:** Testing on the current subsite is aborted due to the failed result.
- **NOABORT:** There were no failed results; therefore, no abort action was taken.

If you want to take some action based on the abort level, you can do it at the `UAP_HANDLE_ABORT`. For example, if you want to take different actions at different levels of abort, you can write the following code in a Keithley User Library Tool (KULT) module and run the module at the `UAP_HANDLE_ABORT`.

In the include files section in the KULT parameter window, append:

```
   #include "COM_usrlib.h"
   #include "ktxe_types.h"
In the KULT main window, type code similar to the following:
{
   int abort;
   /* get the abort flag from the data pool */
   abort = *(int *) dpGetPointer("abort_level", INT_P);
   /* take action based on the abort flag */
   switch(abort)
          {
          case LOTABORT:
                 {
                 /* User Lot abort code */
                 break;
                 }
          case WAFERABORT:
                 {
                 /* User Wafer abort code */
                 break;
                 }
          case SITEABORT:
                 {
                 /* User Site abort code */
                 break;
                 }
          case SUBSITEABORT:
                 {
                 /* User Subsite abort code */
                 break;
                 }
          default:
                 {
                 /* default code */
                 }
          }
}
```

Failed results linked list

The failed results are logged to the event log by default. You can enable event logging by using the `-ev` flag at the command line when starting the Keithley Test Execution Engine (KTXE). The logging of the failed results in the event log will be of the following format:

```
"ABORT REASON id = ... limit high = ... low = ... value = ..."
```

If the failed results information in the event log is not sufficient, you also have access to the failed results linked list at `UAP_HANDLE_ABORT`. The structure for a node of the linked list (defined in `ktxe_types.h`) is:

```
typedef struct _failed_result_list
{
char  id[PARAM_ID_LENGTH];  /* failed result name */
float  high;                 /* high limit */
float  low;                  /* low limit */
char   abortflag[LIMIT_ABORTSTR_LENGTH];
/* abort level - LOT, WAFER, etc. */
   char   abortfield[LIMIT_ABORTSTR_LENGTH];
/* abort limit - Spec, Ctrl, etc. */
   float  value; /* failed result value */
   struct _failed_result_list *next;
/* pointer to the next failed result */
}
failed_result_list_t;
```

The following is an example Keithley User Library Tool (KULT) module that you can create and run at `UAP_HANDLE_ABORT`. The code below traverses through the failed results linked list to find out if a specific result failed and takes the appropriate action in such a situation.

In the include files section in the KULT parameter window, append:

```
#include "COM_usrlib.h"
#include "ktxe_types.h"
```

In the KULT main window, type code similar to the following:

```
{
 failed_result_list_t *failed_result_ptr;
 /* get the pointer to the beginning of the list from the
   data pool */
 failed_result_ptr = (failed_result_list_t*)
dpGetPointer("failed_result_list", LONG_P);
 /* Traverse through the linked list to find the required result name */
 while(failed_result_ptr != NULL)
 {
   if(!strcmp(failed_result_ptr->id, "MyResult"))
   {
   /* found the required result in the linked list */
 /* Take the appropriate actions -- USER CODE*/
   break;
  }
  failed_result_ptr = failed_result_ptr->next;
   /* Required result not found yet, go to the next node of the linked list */
 }
 if(failed_result_ptr == NULL)
 {
  /* "MyResult" was not in the failed result list */
  /* Take any default action if required -- USER CODE */
 }
}
```

The following are notes about the abort action flag and the failed results linked list:

- The user access point `UAP_HANDLE_ABORT` is encountered after the execution of every test (for example, a Keithley test module). Therefore, the abort flag and the failed results list are created for each test.
- Both the abort action flag and the failed results list are available only at `UAP_HANDLE_ABORT`. They are put into the data pool after `UAP_TEST_DATA_LOG` and are removed from the data pool after `UAP_HANDLE_ABORT`. If you want to preserve the values of the abort flag or the failed results list, you must create your own working copy at `UAP_HANDLE_ABORT`.
- The abort action flag represents the highest level of abort action in the list of failed results. For example, if you have a list of five failed results and four of them have an abort level of SUBSITE and one has an abort level of WAFER, the abort flag will be set to WAFERABORT.
- If there were no failed results generated from the execution of a test, the `failed_result_list` in the data pool will be NULL.

Refer to Limits File Editor (LFE) (on page 6-78) for more information on creating the limits files and setting the abort actions and abort limits for results.

KTXE results and their structures

The result structures and results are available after an execution of a Keithley test module (KTM) by the Keithley Test Execution Engine (KTXE). The result structures are:

- **result_list_t:** results from KTMs
- **failed_result_list_t:** failed results from a KTM

The result variables are:

- **result_list:** results from KTMs
- **failed_result_list:** failed results from a KTM

After execution of a KTM, the `result_list` contains a list of all results. These results in the `result_list` are available at `UAP_TEST_END` and `UAP_TEST_DATA_LOG`. The `result_list` is also available at `UAP_HANDLE_ABORT` if a result failed to pass the check against the limits.

The `failed_result_list` contains only the results that failed. These results failed to pass the check against the limits. The `failed_result_list` is available at `UAP_HANDLE_ABORT`. The structures listed below are located in `ktxe_types.h` with definitions in `ktxe_defs.h`.

result_list

```
typedef struct _result_list
{
   char   id[PARAM_ID_LENGTH];
   float  value;
   int    log;   /* True or False Flag to determine if the result is logged*/
   int    user;  /* True or False Flag for User Data
   Logging */
   struct _result_list  *next;
}
result_list_t;
```

failed_result_list

```
typedef struct _failed_result_list
{
   char   id[PARAM_ID_LENGTH];
   float  high;
   float  low;
   char   abortflag[LIMIT_ABORTSTR_LENGTH];
   char   abortfield(LIMIT_ABORTSTR_LENGTH];
   float  value;
   struct _failed_result_list     *next;
}
failed_result_list_t;
```

Data analysis

The following topics describe the tools required to analyze results data once testing has been completed.

Data output formats

There are two different ways to view test results from within the Keithley Test Environment (KTE) software system:

- **Keithley Summary Utility (KSU)**: This tool is used to review results data in a tabular format.
- **Keithley Curve Analysis Tool (KCAT)**: This tool can be used to plot results data that is generated from a test run within KITT in a graph format.

Each of these programs uses different parameters to produce results data output.

Also included is a conversion utility to export data from the Keithley data file (`.kdf`) format:

- KDFtoKCS File Conversion Utility

Preview

This section describes in detail the Keithley Summary Utility (KSU) interface including menu options, controls, and report formats. KSU takes measurement data from a Keithley data file (`.kdf`) and applies limits from a Keithley limits file (`.klf`) to create a report with summary statistics.

Keithley Summary Utility (KSU)

The Keithley Summary Utility (KSU) takes parameter information and measured data from a Keithley data file (`.kdf`) and applies limits from a Keithley limits file (`.klf`) to create a report with summary statistics.

The KSU utility provides you with the capability to:

- **Generate two types of reports:** A standard report and a raw report. Reports can include raw reporting of all measurements recorded and a summarized report that includes means, standard deviations, and ranges.
- **Export data:** In delimited format to most databases and spreadsheets.
- **Search:** Use the browser utility to search for lots based on a series of search criteria.

All of these features are provided in an easy-to-use graphical user interface.

Summary reports

Keithley Summary Utility (KSU) provides two report forms, standard and raw. The standard report provides a table of tests indexed with a series of data from each test to include mean, minimum, and maximum. The report is preceded by a report header similar to the one shown in the following figure. The report header displays information about the lot, process, and device. The header lines are displayed on each report page.

Figure 82: Sample standard report header

```
                                Lot Summary Report KDF V1.0                                    Page:  1
Lot     : 149616                                Operator   : witzke
Process :                                       Starttime  : 15-Feb-97 16:14
Device  :                                       System     : 400ux
Testname: testengine                            Teststation: 1
Limits File:LimFile3                            Wafer File:
? = outside valid limits
* = outside selected limits
# = critical result is outside selected limits
Wafers: 1
Sites : 3
```

The raw report includes the same header file displayed in the standard report form. It displays the test data in columns and indexes the data by site, which lets you compare test results between the multiple sites. At the end of each column is the data contained in a standard report.

Both the standard and raw reports let you create an export version of the summary file with delimited fields. You can specify any single character field delimiter, enabling the file to be exported to a spreadsheet or database for additional analysis.

If a test is aborted, all data gathered up to and including the last test performed is recorded in the `.kdf` file. This means that the summary reflects the abort data, and out-of-range values are flagged to show that they meet the criteria required to abort, as specified in the limits file.

Standard report

The standard report form shown in the following figure consists of the report header field, included in all but export report forms, followed by the lot data.

Figure 83: Standard report

```
                                Lot Summary Report KDF V1.0                                    Page:  1
Lot     : example                               Operator   : witzke
Process : Test                                  Starttime  : 18-Apr-1997 07:42
Device  : new                                   System     : sawsun
Testname: ktxe                                  Teststation: 1
Limits File:                                    Wafer File:
? = outside valid limits
* = outside selected limits
# = critical result is outside selected limits
Wafers: 1
Sites : 1

Name                          Units       Mean      SDEV      Min       Max      %SDEV  SpecL     SpecH     %Spec  CNT %Vld
-------------------------------------------------------------------------------------------------------------------------
sice                                    1.55e-10  0.00e+00  1.55e-10  1.55e-10   0.0 -1.00e+15  1.00e+15  50.0*    1  50.0*
sbeta                                   0.00e+00  0.00e+00  0.00e+00  0.00e+00   0.0 -1.00e+15  1.00e+15   0.0*    0   0.0*
sbmax                                  -5.00e+14  7.07e+14 -1.00e+15  1.11e+09 141.4 -1.00e+15  1.00e+15 100.0     2 100.0
sicmax                                  8.44e-20  1.19e-19  1.40e-45  1.69e-19 141.4 -1.00e+15  1.00e+15 100.0     2 100.0
tee_test_fg                             2.00e+00  1.00e+00  1.00e+00  3.00e+00  50.0 -1.00e+15  1.00e+15 100.0     3 100.0
```

The header is placed at the top of each page of the summary report. An empty field in the header indicates the corresponding field in the data file was empty at the time of the search. The data is listed with test names as row labels on the left, and test information in columns across the top. The test information consists of the mean, minimum, and maximum for the sites based on the test performed.

Following the header is a series of column field labels. These fields contain data derived from the `.kdf` file. Some fields such as %Spec and %Vld include an asterisk next to the data points. This asterisk indicates the data is out of range and alerts you to potentially bad data points. The following paragraphs describe the fields that are the column indexes for the standard report shown in the previous figure.

If a result is outside user-specified limits (specification limits by default) and the critical flag in the Limits File Editor was set to true, a `#` is displayed on the report, and the result is not used in any calculations.

Name

This is the parameter name to which the data corresponds. The Name field is the row index of the summary report.

Units

This is the data units specified in the limits file. This field is blank if no limits file is specified.

SpecL and SpecH

Specification constraints from the limits file. These values let you quickly identify data points that are out of range. If no limits file is specified, they default to -1.00e+15 and 1.00e+15, respectively.

Mean

The mean for each parameter is calculated from the data of all test sites. It is the arithmetic mean of all the results for the wafer and the specified parameter.

Min and Max

Minimum and maximum for the data points of the parameter.

SDEV

This is the standard deviation from the mean of the data points of the parameter.

%SDEV

The standard deviation expressed as a percentage plus or minus from the mean.

CNT

The number of data points used in calculations.

%Vld

The percentage of data points that are within the valid range specified in the limits file. This is 100% if no limits file was specified.

%Spec

The percentage of data points that are within SpecL and SpecH.

Raw report

The raw report is an expanded version of the standard report. The additional data found in the raw report consists of the actual data points derived from the `.kdf` file.

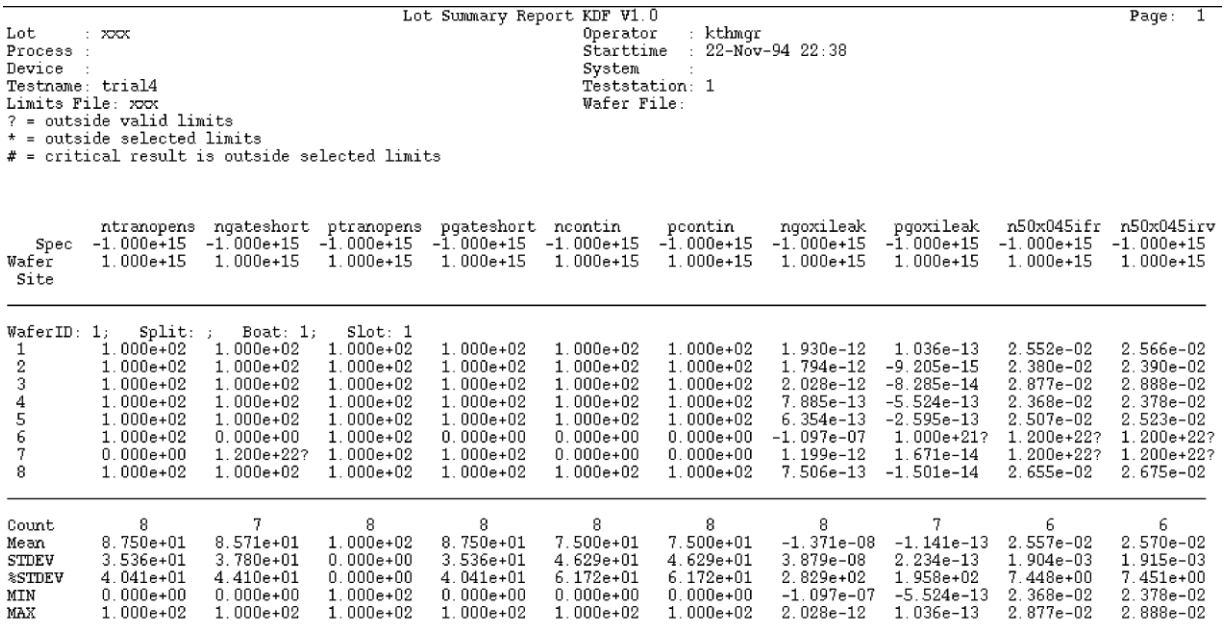
The raw report uses the same header file as the standard report. In a raw report, the parameters are listed in columns instead of rows. The information found in a standard report is listed in the bottom rows of each raw report page.

This information is calculated in the same manner as a standard report. In rows above the standard report data are the wafer and site names for each data point. A specific data point for a test is found as follows:

1. Find the appropriate wafer and site in the rows along the left side of the summary report.
2. Read across the row until the appropriate parameter or test is found. This value is the data point.
3. The relative information for the other data points in the parameter or test can be found by reading down the column until the mean, minimum, and maximum are found.

The following figure is a raw report generated from the same `.kdf` file as the standard report shown in the previous figure.

Figure 84: Raw report



Lot Summary Report KDF V1.0 Page: 1

Lot : xxx
Process :
Device :
Testname: trial4
Limits File: xxx
Operator : kthmgr
Starttime : 22-Nov-94 22:38
System :
Teststation: 1
Wafer File:

? = outside valid limits
* = outside selected limits
# = critical result is outside selected limits

	ntranopens	ngateshort	ptranopens	pgateshort	ncontin	pcontin	ngoxileak	pgoxileak	n50x045ifr	n50x045irv
Spec	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15	-1.000e+15
Wafer	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15	1.000e+15
Site										
WaferID: 1; Split: ; Boat: 1; Slot: 1										
1	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.930e-12	1.036e-13	2.552e-02	2.566e-02
2	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.794e-12	-9.205e-15	2.380e-02	2.390e-02
3	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	2.028e-12	-8.285e-14	2.877e-02	2.888e-02
4	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	7.885e-13	-5.524e-13	2.368e-02	2.378e-02
5	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	6.354e-13	-2.595e-13	2.507e-02	2.523e-02
6	1.000e+02	0.000e+00	1.000e+02	0.000e+00	0.000e+00	0.000e+00	-1.097e-07	1.000e+21?	1.200e+22?	1.200e+22?
7	0.000e+00	1.200e+22?	1.000e+02	1.000e+02	0.000e+00	0.000e+00	1.199e-12	1.671e-14	1.200e+22?	1.200e+22?
8	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	7.506e-13	-1.501e-14	2.655e-02	2.675e-02
Count	8	7	8	8	8	8	8	7	6	6
Mean	8.750e+01	8.571e+01	1.000e+02	8.750e+01	7.500e+01	7.500e+01	-1.371e-08	-1.141e-13	2.557e-02	2.570e-02
STDEV	3.536e+01	3.780e+01	0.000e+00	3.536e+01	4.629e+01	4.629e+01	3.879e-08	2.234e-13	1.904e-03	1.915e-03
%STDEV	4.041e+01	4.410e+01	0.000e+00	4.041e+01	6.172e+01	6.172e+01	2.829e+02	1.958e+02	7.448e+00	7.451e+00
MIN	0.000e+00	0.000e+00	1.000e+02	0.000e+00	0.000e+00	0.000e+00	-1.097e-07	-5.524e-13	2.368e-02	2.378e-02
MAX	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	1.000e+02	2.028e-12	1.036e-13	2.877e-02	2.888e-02

KSU description

The Keithley Summary Utility (KSU) graphical user interface lets you choose one of several lots to use in summary report creation. This section provides detailed information and procedures for using the KSU graphical user interface and KSU tools.

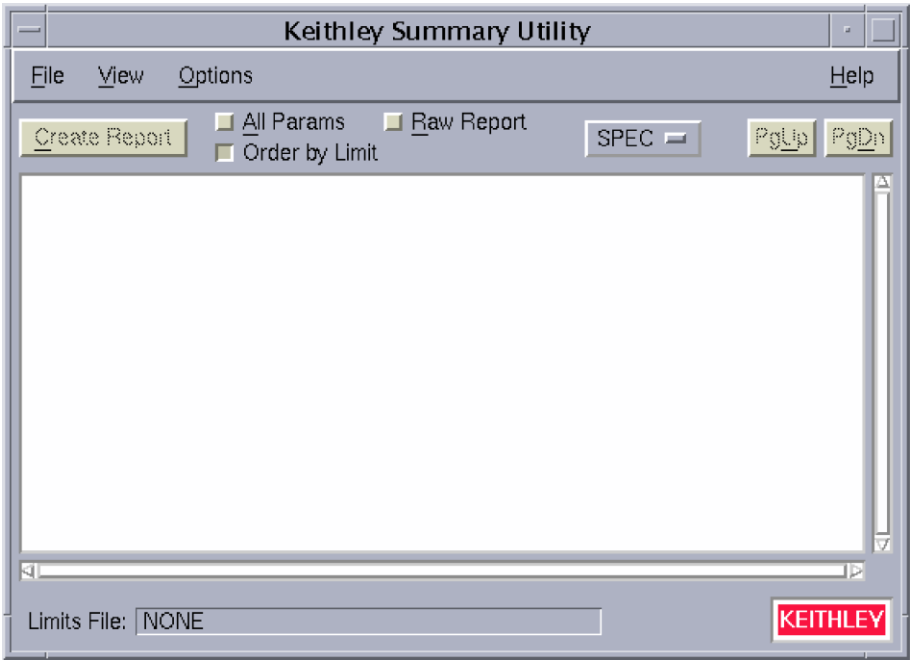
Starting KSU

The Keithley Summary Tool (KSU) can be started from the KSU icon or the command line. Some examples are:

- Double-click the KSU icon or the executable file.
- Click the KSU icon or the executable file and select **Run**.
- Enter the executable file at a command line prompt, and press **Enter**.

After activating KSU, the KSU main window, shown in the following figure is displayed.

Figure 85: KSU main window



KSU main window

The Keithley Summary Utility (KSU) main window menu bar contains the following selections:

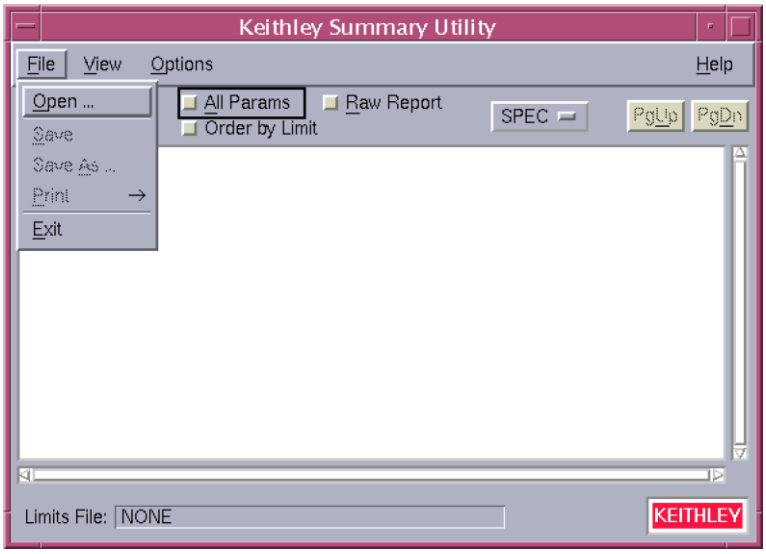
- File menu
- View menu
- Options menu
- Help menu

File menu

Selecting **File** produces the menu shown in the following figure. The menu items are:

- **Open:** Produces the Keithley Summary Utility (KSU) Lot Browser window.
- **Save:** Saves the presently displayed summary file using the present name.
- **Save As:** Saves the presently displayed summary file using a new name.
- **Print:** Produces the print cascade menu, allowing you to choose print options.
- **Exit:** Exits KSU.

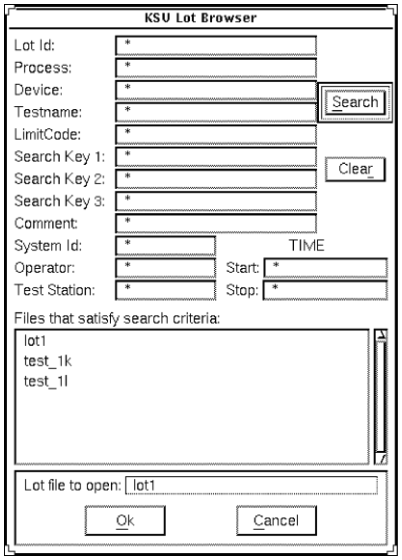
Figure 86: KSU file menu



KSU lot browser window

The Lot Browser window, shown in the following figure, is displayed when you select **Open** from the **File** menu.

Figure 87: KSU lot browser window



The lot browser allows data entry in all of the provided fields. These fields are defined in the `.kdf` file, and only those defined are available to the lot browser search fields. Undefined fields appear blank.

The lot browser search fields support the asterisk (`*`) and question mark (`?`) standard wildcard characters. The asterisk (`*`) is used as a default character in any search field left blank. When the browser is first opened, it performs a search using the standard wildcards. The following table contains a list of valid search criteria and a list of lots displayed. In the example, the possible lots are test 1, test 1-1, test 2, testing, testit, and best1.

Sample search criteria

Search criteria	Lots displayed
*	test1, test1-1, test2, testing, testit, best1
test*	test1, test1-1, test2, testing, testit
test?	test1, test2
test1*	test1, test1-1
?est1	best1, test1
test??	testit

The lot browser's main function is to search the `.kdf` file and produce reports based on that search.

Lot browser controls

The lot browser functions are controlled with following screen buttons:

- **Search**: Initiates a new lot file search based on the present search criteria. A search is initiated the first time in a session you access through the Lot Browser window from the Open selection of the File menu. Each time the Lot Browser window is accessed after that, you must press this button to initiate a search.
- **Clear**: Clears all search criteria and returns each field to the `*` default.
- **OK**: Closes the Keithley Summary Utility (KSU) Lot Browser window and causes the selected lot file to transfer to the KSU main window for report generation.
- **Cancel**: Exits the KSU Lot Browser window and returns to the KSU main window.

The Search results are displayed in the Files that satisfy search criteria list box. To select a lot for search and report generation, click the lot and click the **OK** button, or double-click the lot.

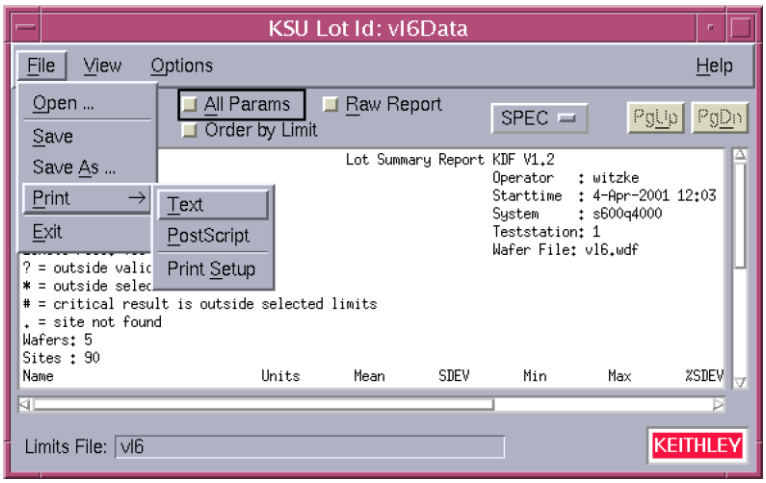
The Lot Browser window closes and the selected lot is transferred to the KSU window for processing. Selecting the **Cancel** button removes the Lot Browser window from the screen and returns control to the KSU main window.

Print menu

When the **Print** option in the **File** menu is selected, the menu shown in the following figure is displayed with the following options:

- **Text**: Prints to non-postscript printers according to formatting data in the print setup.
- **PostScript**: Prints to postscript printers according to formatting data in the print setup.
- **Print Setup**: Sets up the page format for printing.

Figure 88: KSU print menu

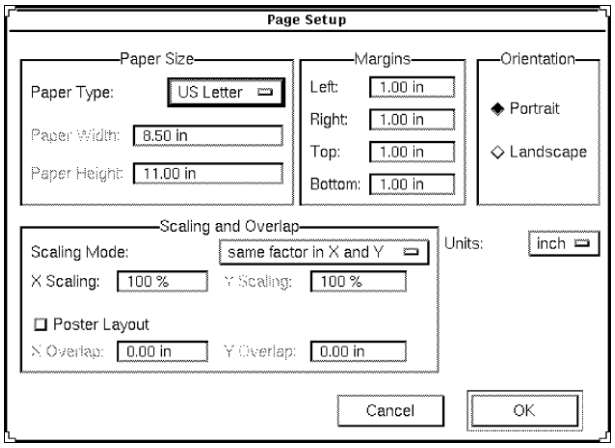


Print Setup

Use the Print Setup selection to set up the page format:

1. Select Print Setup from the Print menu. The Page Setup window shown in the following figure is displayed.

Figure 89: Page setup window



2. Select the page options:
   - Paper Type
   - Paper Width
   - Paper Height

NOTE

The width and height of the paper can only be changed if a paper type of **Other** is selected.

   - Margins
   - Orientation (portrait or landscape)

NOTE

Landscape prints sideways on a printer. Make sure your printer will support this option before printing.

   - Scaling and Overlap. Scaling Mode selections include:
     - **Same factor in X and Y:** The X and Y axes are changed by the same percentage.
     - **Separate X and Y factors:** The X and Y axes are changed independently.
     - **Fit to page:** The X and Y axes are changed accordingly to fit the paper size selected.
     - **Fill one page (unequal):** The data is sized to fill one page, and the X and Y axes will be unequal.
     - **Fit page width:** The data is sized to fit the width of the paper size selected.
     - **Fit page length:** The data is sized to fit the length of the paper size selected.

   If the data area is too large to fit the width and height of the paper, select **Poster Layout**. The X and Y overlap determines how much data is repeated on each sheet so the sheets can be pasted together.
   - Units (inch, mm, or point)
3. Click **OK** to accept the page setup or **Cancel** to abort the printer setup operation.

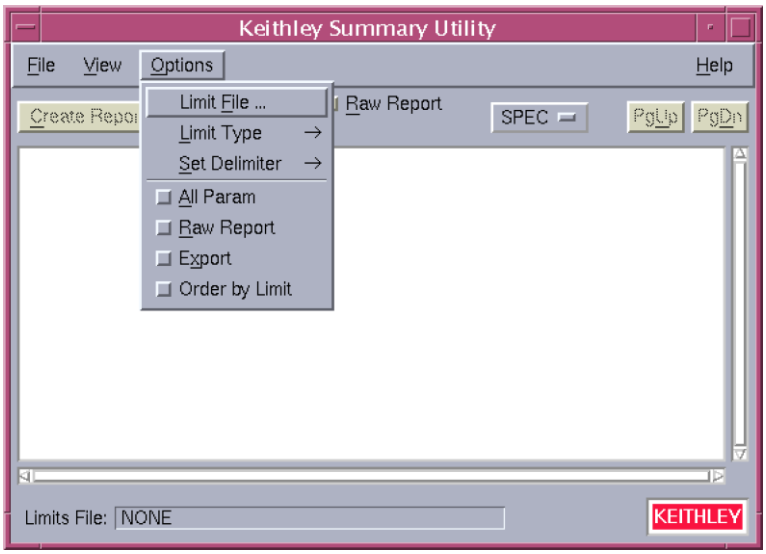
View menu

Selecting **View** lets you select the display format of the Keithley Summary Utility window.

Options menu

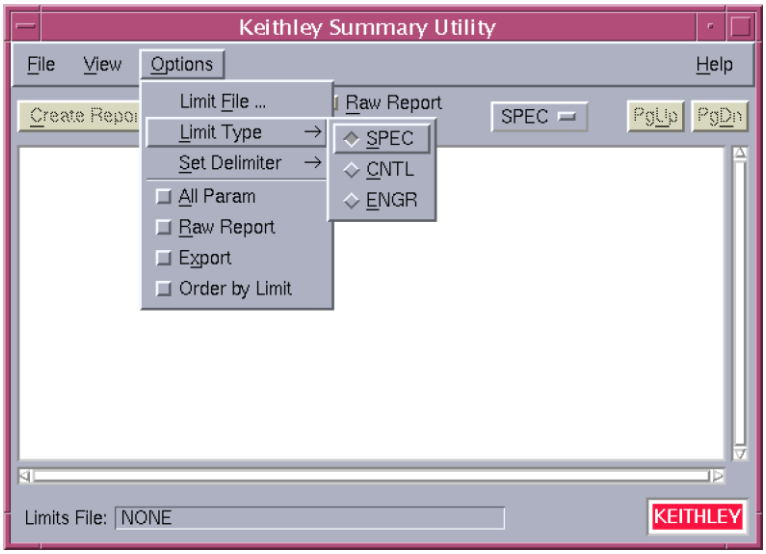
The Options menu shown in the following figure contains the following selections:

Figure 90: KSU options menu



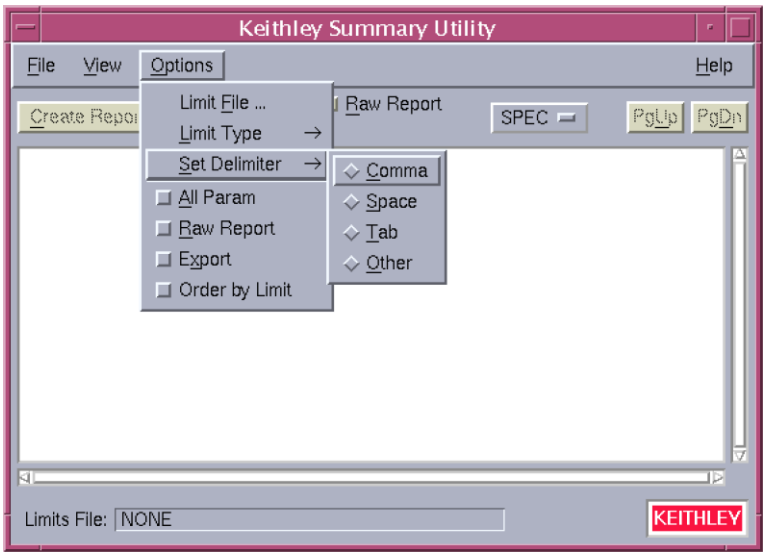
- **Limit File:** Select a limit file to apply to the Keithley Summary Utility (KSU).
- **Limit Type:** Choose specific limits from the chosen limits file (see the following figure).

Figure 91: Limit type menu



- **Set Delimiter:** Choose a delimiter to apply when the KSU file is exported using the Export option.

Figure 92: Set delimiter menu



- **All Param:** Include all available parameters in the KSU report.
- **Raw Report:** Access all information contained in the selected lot file.
- **Export:** Export the KSU report with the delimiter chosen using the Set Delimiter option.
- **Order by Limit:** Create report results in the order specified in the limits file.

Help menu

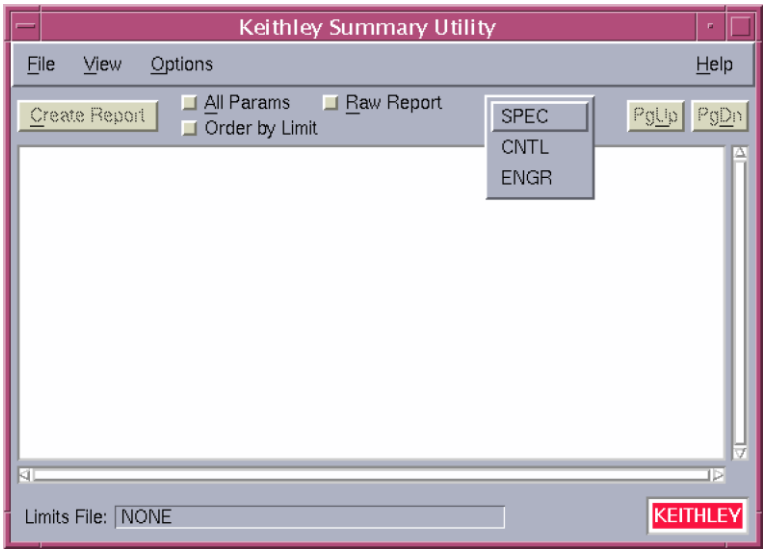
The Help menu item contains the following selections:

- **KSU Documentation:** Provides help information on selected topics about the Keithley Summary Utility (KSU).
- **About:** Displays the installed KSU version number.

Other KSU main window controls

The Keithley Summary Utility (KSU) main window contains the following control selections.

Figure 93: Control selections



- **Create Report:** Initiate KSU report generation using the selected parameters.
- **All Params:** Include all available data file information in the KSU report.
- **Order by Limit:** Direct KSU to produce its report in the same manner as the `-e` command-line switch.
- **Raw Report:** Direct KSU to produce its report in the same manner as the `-r` command-line switch.
- **SPEC, CNTL, ENGR:** Choose which parameter set to apply to KSU report generation.
- **PgUp and PgDn:** Scroll through the displayed report one page at a time.
- **Scroll Bars:** Position the displayed file within the limits of a single displayed screen using the scroll bars on right side and bottom of the window.
- **Limits File:** Display the presently selected limits file.

Command-line options for KSU

The following list contains the command-line options that can be entered when starting the Keithley Summary Utility (KSU).

`-a` Report on all parameters, even if they are not in the limits file.

`-c` Disable the graphical user interface (GUI) (use the lotsummary interface).

`-e` Order results as specified in the limits file.

`-f<filename>` Specify the limits file to use. Default is limits file within lot.

`-d<char>` Set the delimiter. Default is a space, except for the Export version, which is a comma.

`-h` Display the help information.

`-l<char>` Specify which limits to use in addition to the valid limits where *<char>* is `s` = spec (default), `c` = control, *e* = engineering.

`-n` Do not print report header.

`-o<filename>` Set the output filename. Default is `<lotname>.sum`.

`-p` Send the output to a printer. Default is no print.

`-r` Generate a raw report. Default is Lot Level Summary Report.

`-s` Send the output to the screen. Default is no screen.

`-x` Generate an export version of the report. Default is no export.

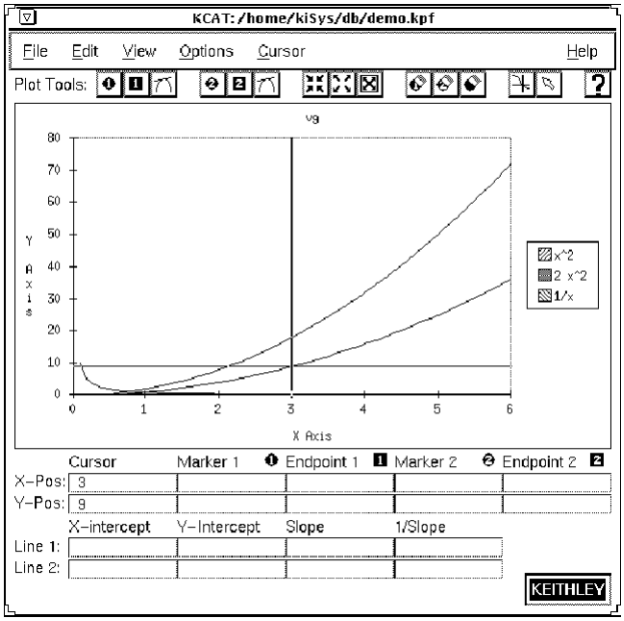
Keithley Curve Analysis Tool (KCAT)

The following topics describe the windows and features associated with the Keithley Curve Analysis Tool (KCAT). The main function of KCAT is to graph results data and calculate slopes, intercepts, and tangents on the resulting curves.

KCAT main window

The following figure shows the Keithley Curve Analysis Tool (KCAT) main window, which is the primary interface to the graphing capabilities of KCAT. When KCAT is executed, the main window is displayed in the upper left quadrant of the screen.

Figure 94: KCAT main window



The main window has the following sections:

- Menu bar
- Tool bar
- Graph area
- Data fields

Menu bar

The menu bar contains the following menus:

File: Load, save, or print Keithley plot files, and exit the Keithley Curve Analysis Tool (KCAT).

Edit: Customize the X and Y axis labels.

View: Display the full KCAT main window, show the KCAT Data window, and the KCAT Scaling window.

Options: Select color or monochrome graphing, line or scatter graph, and the marker type.

Cursor: Customize how the cursor interacts with the graphing area.

Help: Displays KCAT documentation and the KCAT software revision level.

Tool bar

The tool bar contains plotting tools that let you:

- Place and clear markers and endpoints on the graph.
- Draw tangents.
- Zoom in, zoom out, and autoscale the viewing area of the graph.
- Change the interaction of the cursor with the graph.
- Access online help documentation.

Graph area

The graph area contains the following:

- The plotted graph.
- The graph label.
- The graph legend.
- The X and Y axis labels.

Data fields

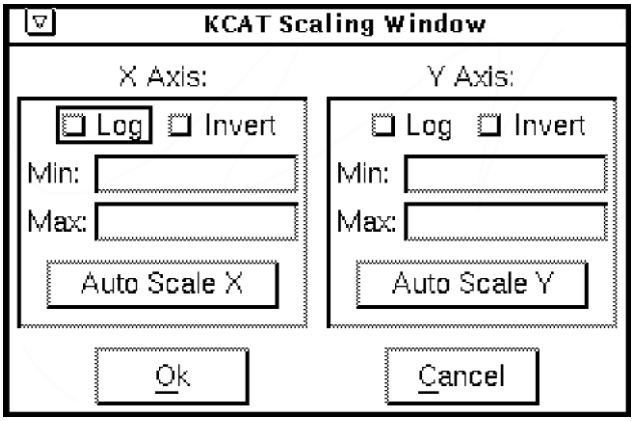
The data fields display information about the graph. The status line gives a description of the present menu or tool being used.

Scaling window

The Scaling window, shown in the following figure, lets you manipulate the graph by turning selecting the following options or scaling the range of the axes.

- **Log:** Change either the X axis data or the Y axis data to the logarithmic equivalent.
- **Invert:** Invert either the X axis data or the Y axis data.

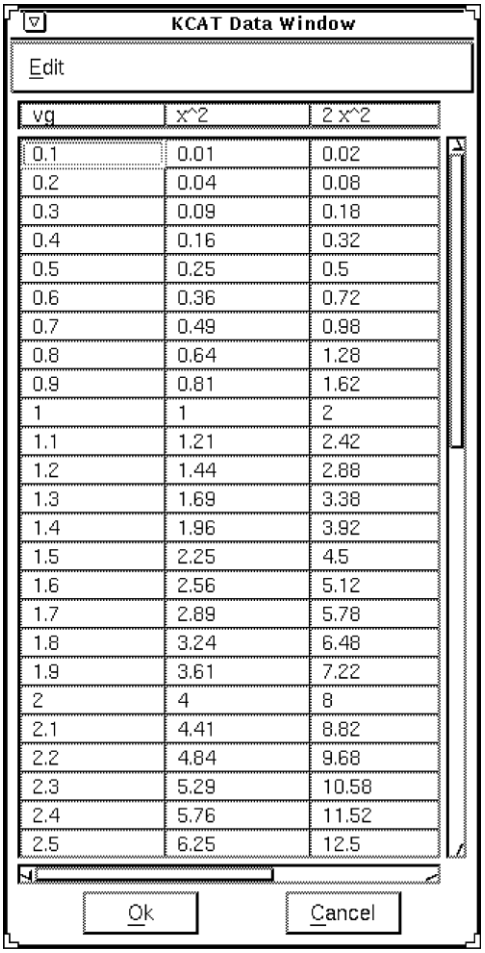
Figure 95: Scaling window



Data window

The Data window contains a listing of all the data points associated with the present graph.

Figure 96: Data window



Pop-up tools menu

The pop-up tools menu contains a listing of all of the tools in the tool bar. It is accessed by pressing the center mouse button when the cursor is anywhere over the graph area.

KDFtoKCS File Conversion Utility

This utility reads a specified Keithley Data File (`.kdf`) and writes an equivalent Cornerstone format file (`.kcs`). Both are ASCII comma-delimited data files. This utility is useful if you want to use a spreadsheet or other Cornerstone-compatible document reader to interpret and edit data.

Syntax of command:

```
KDFtoKCS [-c category][-f substitution_string][-h]
     [-l limit_file][-o outfile][-p] kdf
```

Where:

- `category` = The limits category of interest
- `substitution_string` = An optional string to be substituted into the `.kcs` file wherever the data value is invalid
- `limit_file` = The name of an appropriate Keithley Limits File (`.klf`)
- `outfile` = The name of the intended output `.kcs` file
- `kdf` = The name of the `.kdf` to be converted

The program first accesses the Keithley Limits File (as specified in the `.kdf` file or by using the `-l` command-line option) to write the header row to the `.kcs` file. There are several ways to manipulate this header row, which lists the names and units of the data parameters (see the command-line options list below). The header row has the following format:

```
Wafer,Site,Col,Row,Param1.name Param1.units,Param2.name
     Param2.units,…,ParamN.name ParamN.units
```

The program then proceeds to write each data row containing the site-specific data defined in the header row. The data rows have the following format:

```
WaferID,SiteID,ColNum,RowNum,Param1.val,Param2.val,…,ParamN.val
```

When finished, the output `.kcs` file is located in a folder specified by the environment variable `$KIDB` and has the name of the lot as defined in the `.kdf` file (or, alternately, the name supplied using the `-o` option).

The command-line options are as follows:

- `-conly` = The parameters of the specified category are listed. Only one category may be specified with this option, and it must match at least one category of the parameter exactly, or else the parameter will be discarded.

NOTE

Any given parameter may have several categories if they are comma or whitespace-delimited within the category field of their entry in the `.klf` file.

- `-fthe` = Program substitutes the string argument for data values that do not lie within the valid limits (or omits these values if no substitution string is supplied).
- `-hdisplays` = A help file on the screen.
- `-lsets` = The limits file name.
- `-osets` = The output `.kcs` file name.
- `-puses` = The parameters as they appear in the `.klf` file (default is to get the parameters from the first site entry in the `.kdf` file).

Example

Given the following `.kdf` and `.klf` files:

4 lotData1.kdf

```
TYP,KDF V1.0
LOT,lotData1
TST,ktxe
SYS,Q22
TSN,1
OPR,operator1
LMT,detailLimit
STT,01-Jan-2017 13:26
<EOH>
Wafer_01,,1,1
1,-4,1
icur,1.3124e-10
<EOS>
Wafer_01,,1,1
1,-4,2
icur,1.3126e-10
<EOS>
<EOW>
```

detailLimit.klf

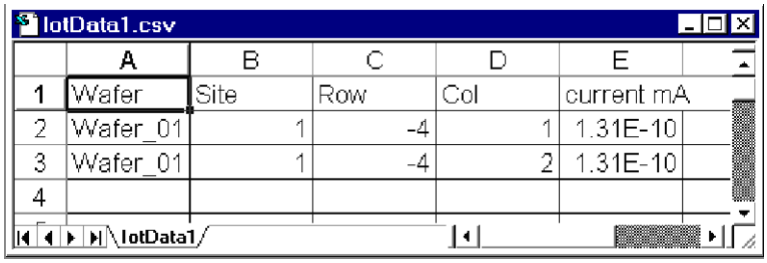
```
#Keithley Parameter Limits File
Version,1.0
File,/TestArea/detailLimit.klf
Date,01/01/2018
Comment,KLF
<EOH>
ID,icur
NAM,current
UNT,mA
RPT,1
CRT,0
TAR,0.0000e+00
AF,0
AL,0
VAL,1.3100e-10,1.3125e-10
SPC,-2.0000e-10,2.0000e-10
CNT,-2.0000e-10,2.0000e-10
ENG,-2.0000e-10,2.0000e-10
<EOL>
```

5 lotData1.kcs

```
Wafer,Site,Row,Col,current mA
Wafer_01,1,-4,1,1.3124e-10
Wafer_01,1,-4,2,1.3126e-10
```

When this file is viewed in a spreadsheet program such as Microsoft® Excel®, it appears like this:

Figure 97: Lotdata1.kcs in a spreadsheet



NOTE

The extension was changed from `.kcs` to `.csv` so that Excel can recognize the format.

System administration

The following topics discuss setting up and using the Keithley Test Environment (KTE) software for use with the KTE system. Additional information on system administration is available in the *S540 Administrative Guide* (part number S540-924-01).

System configuration: kth.ini file

The Keithley Test Environment (KTE) system uses a system-wide initialization file called `kth.ini` to configure some of the software components. The format for each entry within the `kth.ini` file is:

```
<COMPONENT>
item = value
```

The definition for each of these parameters is:

- `<COMPONENT>` = One of the KTE software subsystems.
- `item` = The customizable parameter for the subsystem.
- `value` = The value associated with the item.

When started, Keithley application programs search the following locations for the `kth.ini` file:

- The current working directory.
- The login directory.
- The master version in `$KIHOME`.

The following is an example of a `kth.ini` file:

```
<KDF>
#Datapath=$KIDB
Error_Log=/$KIHOME/log/kdf_error.log
<SUM>
Lprint=lpr
LinesPerPage=55
ParamsPerLine=10
<GPIB232CT-UNIT#1>
BAUD=9600
TIMEOUT=10
TTYDEV=/dev/ttya
```

Any parameters defined by the `kth.ini` file can be changed by editing the file with an ASCII editor. The new values are used the next time the program executes.

When you make changes that you do not want other users to be affected by, copy the `kth.ini` file from `$KIHOME` to your present or login directory and edit it there. For example, assume you want to modify the Summary Report Generator `<SUM>` to print five parameters per line, rather than its default setting of 10 per line. Do the following:

1. Copy the `kth.ini` file to your working directory:

   ```
   prompt> cp $KIHOME/kth.ini
   ```

2. Edit the file with text edit or any other ASCII editor.

3. Modify the `<SUM>` parameter `ParamsPerLine` as needed. Do not use spaces around the = sign.

NOTE

If you make an invalid change, for example `ParamsPerLine=-10`, the software ignores the setting and uses a reasonable default value when it executes.

4. Generate your summary report. The Summary Report Generator uses the new value of `ParamsPerLine` if you execute it from the working directory.

To make this change universal, replace the file in `$KIHOME` with the new version.

kptm.ini file

The `ktpm.ini` file allows you to set the user access points (UAPs) that are associated with each engine used in the Keithley Test Program Manager (KTPM). The `ktpm.ini` file, shown below, is located in the `$KI_KTXE_CPF` directory.

When an engine is selected, the UAPs associated with that engine are accessible to the UAP Module/KTM area in KTPM. Only the UAPs listed with the selected engine can appear in this area. Each selection in this field has an associated list of UAPs that become active when the engine version is selected.

```
# Start of File */
<Engine_List>
Engine1 = ktxe,$KIBIN,Initial KI Execution Engine for V5.x
<ktxe>
# process user's ktxe command line arguments
UAP1 = UAP_PROG_ARGS
# start processing the cassette plan
UAP2 = UAP_CASSETTE_LOAD
# start setting up the lot file
UAP3 = UAP_LOT_INFO
# start of prober initialization
UAP4 = UAP_PROBER_INIT
# mismatch detected between cassette plan and prober
UAP5 = UAP_WAFER_MISMATCH
# allow access to wdf before call to PrInit
UAP6 = UAP_ACCESS_WDF_INFO
# called to send init commands to prober
UAP7 = UAP_POST_PROBER_INIT
# before writing lot information
UAP8 = UAP_WRITE_LOT_INFO
# write usertag data after LOT header
UAP9 = UAP_POST_LOT_INFO
# after a wafer load and the wafer is rejected
UAP10 = UAP_WAFERLOAD_STATUS
# error recovery after wafer alignment
UAP11 = UAP_ALIGN_ERROR
# perform AutoZ after first wafer load
UAP12 = UAP_POST_INITIAL_WAFER_LOAD
# start processing next wafer plan
UAP13 = UAP_WAFER_PREPARE
# after OCR and before wafer ID is logged to data file
UAP14 = UAP_VALIDATE_OCR
# start executing wafer plan
UAP15 = UAP_WAFER_BEGIN
# start of next site processing
UAP16 = UAP_SITE_CHANGE
# start of next subsite processing
UAP17 = UAP_SUBSITE_CHANGE
# start of next ktm processing
UAP18 = UAP_TEST_BEGIN
# end of processing a ktm
UAP19 = UAP_TEST_END
```

```
# after test data has been logged
UAP20 = UAP_TEST_DATA_LOG
# processing an abort condition
UAP21 = UAP_HANDLE_ABORT
# end of current sub-site processing
UAP22 = UAP_SUBSITE_END
# end of current site processing
UAP23 = UAP_SITE_END
# end of processing a wafer plan
UAP24 = UAP_WAFER_END
# end of processing the cassette plan file
UAP25 = UAP_LOT_END
# before leaving the execution engine
UAP26 = UAP_ENGINE_EXIT
# called as an atexit function
UAP27 = UAP_ABORT_EXIT_HDLR
# called if prober err and function exists
UAP28 = UAP_PRB_ERR_HDLR
# called when pause/cont is pressed on StatDlg
UAP29 = UAP_STATUS_CHANGE
# before profiling a wafer
UAP30 = UAP_PROFILE_WAFER
# Place probers that support PrSetSlotStatus here in this list.
# The YES is mandatory!!!
<AbsProbers>
EG40=YES
EG2X=YES
TSK9=YES
P8=YES
# This is used for working off-line. Comment this out if you want.
FAKE=YES
/*End of File */
```

The ktpm.ini file searches through the different directories for the data associated with each UAP in the following order:

```
$KI_KTXE_CPF
$KIHOME
$HOME
```

To modify the `ktpm.ini` file, use textedit or any other ASCII text editor program.

Environment variables

The following list contains the environment variables used by the Keithley Test Environment (KTE) tools and execution engines. Environment variables can be set to determine the default location of data files. Environment variables also provide a mechanism to manipulate different sets of data with the same tools. Set the environment variables to the locations you want and start the tool.

Scripts can be developed that set these variables to point to different areas of the file system. For example, a script, `SetProduction`, could be used to set the environment variables to point to the production data file system. Another script, `SetDevelopment`, could be used to set the environment variables to point to a development data file system.

NOTE

If the environment variables are changed after the tool has started execution, you must exit the tool and restart for any changes to take effect.

NOTE

The Wafer Plan Editor (WPE) is part of the Keithley Test Plan Manager (KTPM). There are references to both KTPM and WPE. They are combined into one tool.

KTPM (path is stored with .wpf, .pcf, and .gdf)

- Cassette: Uses `$KI_KTXE_CPF`
- Global: Uses `$KI_KTXE_GDF`
- KTM: Uses `$KI_KTXE_KTM`
- Wafer plan: Uses `$KI_KTXE_WPF`
- Prober Card: Uses `$KI_KTXE_PCF`
- Wafer Desc: Uses `$KI_KTXE_WDF`
- UAP: Uses `$KI_KTXE_UAP`
- SYSTEMAP file: Uses `$KI_KTXE_SYSTEM_AP`

WPE (part of KTPM)

- Open: Uses `$KI_KTXE_WPF`
- Wafer Desc: `Uses $KI_KTXE_WDF`
- Limits: Uses `$KI_KTXE_KLF`
- Probe Card: Uses `$KI_KTXE_PCF`

Keithley Interactive Test Tool (KITT)

- Open: Uses `$KI_KTXE_KTM`
- Wafer Desc: Uses `$KI_KTXE_WDF`
- Global Data: Uses `$KI_KTXE_GDF`
- Probe Card: Uses `$KI_KTXE_PCF`
- Test Structure: Uses `$KI_KTXE_TSF`
- Parameter Set: Uses `$KI_KTXE_PSF`
- Practice Task: Uses `$KIPGM`

Logic File Editor (LFE)

- Open: `$KI_KTXE_KLF`
- Read ID: `Uses $KI_KTXE_KTM`

Wafer Description Utility (WDU)

- Open: Uses `$KI_KTXE_WDF`

Test Structure File Editor (TSE)

- Open: `$KI_KTXE_TSF`

Keithley Curve Analysis Tool (KCAT)

- Open: `$KI_KTXE_KDF`

Parameter Set Editor (PSE)

- Open: `$KI_KTXE_PSF`

Keithley Operator Interface Editor (KOPED)

- Open: Uses `$KIDAT`

Keithley Summary Utility (KSU)

- Open: `$KI_KTXE_KDF`

Keithley Operator Interface (KOP)

- `.ini` file load: Uses `$KIDAT`

The Keithley startup and login files define the environment variables shown in the following tables. It is good programming practice to use these variables whenever possible. This allows you to move the location of the Keithley directory tree without affecting your programs and scripts.

Keithley directory environment variables

Variable	Definition	Meaning
KIHOME	Installation dir (/opt/ki)	KI HOME directory
KIBIN	${KIHOME}/bin	KI binaries directory
KIDAT	${KIHOME}/dat	KI data directory
KIDB	${KIHOME}/db	KI data base
KILIB	${KIHOME}/lib	KI libraries directory
KILOG	${KIHOME}/log	KI log files directory
KITMP	${KIHOME}/tmp	KI temporary directory
KIINCLUDE	${KIHOME}/include	KI include directory
KI_LOCK_LOC	${KIHOME}/lock	KI_lock_file directory
KIPGM	${KIHOME}/pgm	.ktm and .wdf directory

Open interface environment variables

Variable	Definition	Meaning
ND_HOME	${KIHOME}openint	Open interface home directory.
ND_PATH	${ND_HOME}/lib:$KIDAT	Open interface search file path.
ND_LIB	${ND_HOME}/shr	Open interface library subdirectory.
OIT_LOOK	MOTIF	Open interface graphical user interface (GUI) theme.
OIT_PATH	${OIT_HOME}/lib:$KIDAT	
OIT_LIB	${OIT_HOME}/shr	
OIT_HOME	${KIHOME}/openint	

System environment variables

Variable	Definition	Meaning
KI_SYSTEM	System name string	
KI_PRB_CONFIG	${KIDAT}/prbconfig_xxxx.dat	Prober configuration for system
KI_CONFIGURATION	${KIDAT}/acconfig_QMO.ini	Configuration file for system
KI_KUI_CLASSIC	Default state Undefined. Set to 1 to use old-style Keithley User Interface (KUI) windows	Provides a way to use existing KUI.
KI_ENABLE_2010_DISPLAY	Set to 1 to activate the front-panel display of the optional Model 2010 Digital Multimeter (DMM)	
KI_ENABLE_2410_DISPLAY	Set to 1 to activate the front-panel display of the high-voltage Model 2410 source-measure unit (SMU)	
KI_DEFAULT_LIM_MODE	Set to KI_INDICATOR to use the KI_INDICATOR value (7.0e22) instead of the measured value	
KI_NO_GPIB	Set to KI_NO_GPIB to disable prober GPIB communications	Use to disable prober GPIB communications if the GPIB adapter is missing or defective. Can be set in the $KIHOME/.ki_setup file (if the FAKE prober is specified).
KI_ONLY_GND_CHUCK	Set to 1 to automatically connect the CHUCK to GND if CHUCK is unused	Note that the KI_GND_UNUSED environment variable must also be set to get this behavior.
KI_GND_UNUSED	Set to 1 to automatically connect unused pins (including CHUCK) to GND	
KI_MATRIX_PRINT	Set to 1 to enable debug messages from the matrix driver to be displayed	
KI_KELVIN_CHECK	Set to ON to enable or OFF to disable Kelvin check	Undefined means disabled.

Diagnostics environment variables

Variable	Definition	Meaning
KI_PLATFORM	S530, S535, or S540	Platform type.
KI_DIAGTOOLS_CONFIG	${KIHOME}/dat	Configuration file.
KI_DIAGTOOLS_LOG	${KIHOME}/log	Log output file.

User library tool environment variables

Variable	Definition	Meaning
KI_KULT_PATH	${KIHOME}/usrlib	Location of user libraries.

Test plan manager environment variables

Variable	Definition	Meaning
KI_KTXE_CPF	${KIHOME}/plans	Location of cassette plan files.
KI_KTXE_GDF	${KIHOME}/plans	Location of global data files.
KI_KTXE_KDF	${KIHOME}/db	Location of data files.
KI_KTXE_KLF	${KIHOME}/db	Location of limit files.
KI_KTXE_PCF	${KIHOME}/plans	Location of probe card files.
KI_KTXE_PLANS	${KIHOME}/plans	Plans directory.
KI_KTXE_UAP	${KIHOME}/plans	Location of user access points (UAPs).
KI_KTXE_WDF	${KIHOME}/pgm	Location of wafer description files.
KI_KTXE_WPF	${KIHOME}/plans	Location of wafer plan files.
KI_KTXE_KTM	${KIHOME}/pgm	Location of test macros.
KI_KTXE_TSF	${KIHOME}/plans	Location of test structure files.
KI_KTXE_PSF	$KI_KULT_PATH	Location of parameter set files.
KI_LOCALIZE_CFG	Defined by user (optional)	KTXE/KUI localization file.

Log file environment variables

Variable	Definition	Meaning
KI_KTXE_ERROR_LOG	Defined by user	Keithley Test Execution Engine (KTXE) error log file location.
KI_KTXE_EVENT_LOG	Defined by user	KTXE event log file location.
KI_KTXE_DEBUG_LOG	KTXE debug log file location	
KI_PRB_AUDIT_LOG	Defined by user	Prober transaction log file location.
KI_IC_AUDIT_LOG	Undefined	Determines where any IC error messages will be stored.
KI_TRACE_LEVEL	Undefined	Can be set to TRACE, DEBUG, or ERROR. This determines what level of log messages are displayed by the IC process. Default level is DEBUG.
KI_DISABLE_TRACE	Undefined	Disables any IC log messages from being displayed. Note that if the KI_IC_AUDIT_LOG variable is defined, error level messages are logged.
KI_KISA_DEBUG	Set to 1 to enable debug messages from the kisa process.	
KI_KISA_SHOW_LOG	Set to 1 to enable print-type logging in addition to file logging for the kisa process.	
KI_KTXE_STARTUP_MSGS	Set to 1 to enable KTXE process startup messages.	
KI_ALWAYS_SHOW_DIAGS	Set to 1 to enable version control operation debug messages.	
KI_HIDE_UNSUPPORTED_MSGS	Set to 1 to disable the "UNSUPPORTED LPT COMMAND" messages.	

Using log file environment variables

Classification of errors:

- **Prober Errors**: Recover if possible. Notify, user abortable, logable.
- **LPT Errors**: No recovery. Logable, abort on fatal errors.
- **Data Logging Errors**: Notify and abort, logable.
- **Engine Data Errors:** Non-fatal: Notify and continue, logable. Fatal: Notify and abort, logable.

If these environment variables are defined, the respective log messages are placed in the file specified.

NOTE

If the `-e` and `-ev` command-line options for the execution engine have a file name specified, this file name overrides the environment variables.

- `KI_KTXE_ERROR_LOG`**:** Determines the path and filename of the Keithley Test Execution Engine (KTXE) error log file.
- `KI_PRB_AUDIT_LOG`**:** Determines the path and filename of the transaction log file.
- `KI_LPT_ERRORLOG`**:** Determines the path and filename of the Linear Parametric Test Library (LPTLib) error log file.
- `KI_PRB_ERROR_LOG`**:** Determines the path and filename of the prober error log file.
- `KI_PRB_ERRM`**:** Determines the path and filename of the prober error message file.

Example

To log KTXE events and errors, prober errors, and LPTLib errors to same file:

```
prompt>   setenv KI_KTXE_ERROR_LOG /mydir/file.log
prompt>   setenv KI_PRB_AUDIT_LOG /mydir/file.log
prompt>   setenv KI_LPT_ERRORLOG /mydir/file.log
prompt>   setenv KI_PRB_ERROR_LOG /mydir/file.log
prompt>   setenv KI_PRB_ERRM /mydir/file.log
```

Or, start the execution engine as:

```
prompt>   ktxe -cpf cassettePlan -e 2 /mydir/file.log -ev 2 /mydir/file.log
```

This logs KTXE errors to `/mydir/file.log`.

File management

The system disk on the Keithley Test Environment (KTE) system contains the standard operating system partitions `/`, `/user`, and `/home`. All of the KTE software is located under the `/opt/ki` path. The following is a list of all of the directories located under the `/opt/ki/` path:

- `bin`: Keithley executable files and shell scripts.
- `dat`: Keithley and GUI data files.
- `db`: Data files.
- `doc`: KTE Tools online help
- `include`: Keithley C language header files
- `install`: Installation tools
- `lib`: Keithley static and shared libraries
- `lock`: Keithley lock files; to protect against multiple access writes
- `log`: System log files
- `openint`: Open interface files
- `pgm`: Test macro and wafer description files
- `plans`: Test plan files
- `skel`: Default login scripts
- `src`: Source files for some system files
- `tmp`: System temporary scratch directory
- `unsupported`: Unsupported utilities
- `usrlib`: Keithley User Library Tool (KULT) user libraries

Keithley Component Manager (KCM) utility description

Use the Keithley Component Manager (KCM) utility to package and transfer test plans from one location to another. This utility selects all files required by the test plan and places them into a `*.tar` file. Specify the `-u` switch to include all usrlibs in addition to the required files.

Usage

```
kcm -e efile [-u] [-o <fname>]
   kcm -i ifile [-u] [-f]
```

- `-e | -i`: Export or import files (mutually exclusive).
- `ifile`: Name of file to be imported. The file must be generated by the Keithley Component Manager (KCM) utility. Paths are supported.
- `efile`: Name of file to be exported. Non-usrlib files must include file extension. Paths are ignored.
- `-u`: Optional parameter to include dependent usrlibs. Unless specified, usrlibs are not included in the `efile` or installed from the `ifile`. If the file specified is a usrlib, the `-u` switch has no effect. Only the usrlib specified is bundled.
- `-f`: Optional parameter to force overwriting local files and usrlibs without asking for confirmation.
- `-o <fname>`: Optional export parameter to specify output filename. Default output filename is `efile.tar`.

Samples

Using `kcm -e sample.cpf -u` creates a package that contains all Keithley Test Environment (KTE) files necessary to execute the `sample.cpf` cassette plan.

Using `kcm -i sample.cpf -u` installs this package into the current environment.

Project environments

The `.ki_setup` file has been modified to dynamically adjust your environment variables on a project-level basis. This allows you to set up different areas to store, run, and load your product specific files, or separate production and development code.

You can use the `make_project` script to create a new tree containing the following directories:

- `db`
- `pgm`
- `plans`
- `usrlib`

The script prompts for each of these possibilities and creates the directories as necessary. In the case of the usrlib directory, the original `KIHOME/usrlib` directory is copied to the new location.

The following example creates a new project named `dev1Area`, and the tree root of the project is placed in `/kte/ProjectTree`.

```
Prompt> make_project
Usage: make_project tree projname
Prompt> make_project /kte/ProjectTree dev1Area
   Creating Project
   Would you like to create a new db directory? y
   Would you like to create a new usrlib directory? n
   Would you like to create a new plans directory? y
   Would you like to create a new pgm directory? y
   Creating /ki/dat/dev1Area.env
   Creating /ki/dat/dev1Area.set
   Finished Creating Project
Prompt>
```

The creation of a project places a new file, `projectname.env`, in the `KIDAT` directory. The example above creates the file `dev1Area.env`, which is shown below:

```
setenv KIPROJ /kte/ProjectTree/dev1Area
setenv KI_PROJ_DB /kte/ProjectTree/dev1Area/db
setenv KI_PROJ_USRLIB /ki/usrlib
setenv KI_PROJ_PLANS /kte/ProjectTree/dev1Area/plans
setenv KI_PROJ_PGM /kte/ProjectTree/dev1Area/pgm
setenv KI_PROJ_NAME dev1Area.env
```

There are two ways to reset your environment for a different project.

Invoke `select_project`, which will scan the `KIDAT` directory for all possible `.env` files, present an enumerated list, and ask which one you want. You can cancel by entering `0`, or reload the original variables by entering `111`.

```
Prompt> select_project
```

Your choices are:

- Project 1 is `DemoProject.env`
- Project 2 is `Keithley_Orig.env`
- Project 3 is `current.env`
- Project 4 is `dev1Area.env`

```
Enter your selection or 0 to cancel;
       111 to load KI Original
   >>---> 0
Prompt>
```

The process used by the `select_project` script sets the following environment variables:

- `KIPROJ`: Points to the base path of the project tree.
- `KI_PROJ_DB`: Points to path of directory where Keithley Data Files (KDF) and Keithley Logic Files (KLF) are stored.
- `KI_PROJ_USRLIB`: Points to the usrlib directory where Keithley User Library Tool (KULT) libraries are stored.
- `KI_PROJ_PLANS`: Points to directory where `.cpf`, `.wpf`, `.gdf`, and `.pcf` files are stored.
- `KI_PROJ_PGM`: Points to directory where wafer definition files (WDF) and Keithley Test Module (KTM) files are stored.
- `KI_PROJ_NAME`: Project name.

The `.ki_setup` file uses these base variables to configure the remaining Keithley Test Environment (KTE) environment variables.

A graphical user interface (GUI) window is provided in the Keithley Operator Utility (KOP) and Keithley Test Plan (KTP). By changing the project within these tools, all applications started by these tools use the project environment specified.

When you start a new shell or log in, the last project environment saved in `$HOME/current.env` is loaded. If this file does not exist, the system looks for the existence of `$KIDAT/current.env`. If that does not exist, the system takes the Keithley defaults using the tree off of `$KIHOME`.

The `make_project` and `select_project` scripts are located in the `$KIBIN` directory. They are C-Shell scripts and can be modified to add additional environment variables and capabilities. For example, you can add other project specific environment variables for use in project specific KULT functions, as shown below:

Add the following to the `$KIDAT/development_project.env` file:

```
setenv MY_ENV_STRING ">>> development shell"
```

Add the following to the `KIDAT/production_project.env` file:

```
setenv MY_ENV_STRING ">>> production shell"
```

Now a KULT function can use this environment variable and determine the project and adjust behavior based upon this information. For example:

```
printf("This is the %s\n",getenv("MY_ENV_STRING"));
```

Select tester

The following topics describe how to select the tester.

Setup information

A `$KIDAT/kitester.dat` file contains a list of the QMO numbers of the testers to which the workstation is permitted to connect. To allow connection to additional testers, modify the `$KIDAT/kitester.dat` file to include the QMO numbers. You must also create a `$KIDAT/acconfig_QMO#.ini` file for each QMO listed in the `$KIDAT/kitester.dat` file.

The `select_tester` script creates a `.ki_define_config` file in the user's home directory. This file is then sourced if `select_tester` is executed interactively. If `select_tester` is called noninteractively, then you must source `~/.ki_define_config` so that the Keithley Test Environment (KTE) is configured correctly.

select_tester script

The `select_tester` routine enables the operator to specify to which tester the KTE toolset will connect. A `$KIDAT/kitester.dat` file contains a list of the QMO numbers of the testers the workstation is permitted to connect with. This file is accessed by the `select_tester` script and the list of available testers is shown. The operator can then choose the tester from the list, or accept the currently selected tester if shown (sample display).

This script can be executed interactively or integrated into a `.cshrc` file for automatic tester selection at login.

Interactive execution:

```
prompt> source select_tester
```

or

```
prompt> select
```

No parameters are permitted. Tester selections are interactive. `select` is an alias for source `${KIBIN}/select_tester`.

Noninteractive execution:

```
prompt> select_tester 2
```

A parameter is permitted to specify tester, but the `~/.ki_define_config` execution will have no effect.

To use from the `.cshrc` file, include after source `~/.kth_startup`.

Example:

```
source ~/.kth_startup
select_tester 2
source ~/.ki_define_config
```

Sample display

The following example display shows `select_tester` being executed with the operator choosing the current tester, QMO 4001.

```
prompt>source select_tester
-----------------------------------------
Keithley S540 System Selection Utility:
-----------------------------------------
Number    QMO    System Name
----------        --------     -------------------------
   -->      1    - 4001 Tester1Nice Name
            2    - 4002 Tester2Nice Name
            3    - 4003 Tester3Nice Name
           99    - Debug Tester
            0    - Exit
    Enter Number: [1]
Configuring Environment for QMO:4001 - Tester1Nice Name
   prompt>
-------------------------------------------------------------------------------
```

NOTE

In the example above, the system is identified as `S540`; if you have a different system, this number will be different.

Selecting tester 99, the debug tester, will use the fake prober. You do not control an actual tester or prober.

The `select_tester` script creates a `.ki_define_config` file your home directory. This file is then sourced (if in interactive mode) to initialize the proper environment variables based upon the tester selected.

.ki_define_config

The `select_tester` script creates a `.ki_define_config` file your home directory. This file is then sourced (if in interactive mode) to initialize the proper environment variables based upon the tester selected.

S540 environment variables

The following environment variables are defined by `select_tester` that allow the Keithley Test Environment (KTE) toolset to communicate with the proper tester:

```
setenv KI_CONFIGURATION ${KIDAT}acconfig_$QMO.ini
setenv KI_PRB_CONFIG ${KIDAT}prbcnfg_$QMO.dat
setenv KI_SYSTEM $SystemName
```

Where:

`$QMO` = The QMO of the tester selected by the `select_tester` script.

`$SystemName` = The `SystemName` value found in the `$KIDAT/acconfig_QMO#.ini` file.

ki_setup_$QMO

Additional tester-specific setup can occur using a `$KIHOME/ki_setup_$QMO` file. This file, if it exists for a particular QMO number, is also sourced so that additional environment variables or other customizations can be initialized.

kitester.dat

The following sample `kitester.dat` file allows the workstation to connect to testers with QMO numbers 4002, 4003, and 4010.

```
#
# QMO numbers of testers accessible by this workstation account
# Modify this file as necessary to allow/disallow access to testers
# Only the QMO number is necessary.
#
# There MUST be a acconfig_nnnn.ini file for each QMO number listed!!!
#
# example:
#4001
4002
4003
4010
```

System customization

The following topics discuss system customization.

Logging in to the workstation

The Keithley Test Environment (KTE) system ships with four default user accounts installed:

- **System Administrator:** This account is for the person responsible for overall operation of the system and workstation. The system manager typically performs system backups, generates new accounts for users, installs software upgrades, and maintains the Linux® environment.
- **Test System Manager:** This account is for the person responsible for maintaining files on the system. In many environments, the System Administrator and Test System Manager roles are assigned to the same person.
- **Test Engineer/Programmer:** This account is for the engineers writing parametric test plans. This account has the access privileges required to create, modify, and run test plans.
- **Test Station Operator:** This account is for anyone responsible for operating a wafer prober, running test plans, or other application programs.

These four accounts are summarized in the following table.

Type of account	Log-in name	Password	Log-in directory
System Administration	`root`	`keithley`	`/`
Test System Management	`kthmgr`	`kthmgr`	`/export/home/kthmgr`
Test Program Development	`kthprg`	`kthprg`	`/export/home/kthprg`
Operators Login	`kthopr`	`kthopr`	`/export/home/kthopr`

You can customize your system to change the level of access for each type of account by modifying the login directory to limit or expand access as required. You can also change the access password for each account.

Configuring the tool.tpi file

The `tool.tpi` file is the main configuration for the Keithley Tool Palette (KTP) and contains the names of all of the other files needed to access the tools available from the KTP.

The `tool.tpi` file has four major categories to be defined:

- Keithley-Tools
- Unix-Tools
- Unix Administration
- User Programs

These categories correspond to the icon sets available in the KTP. All four categories use the same configuration scheme.

The first line of each category within the `tool.tpi` file is defined by the category name enclosed by less than (<) and greater than (>) symbols. The following paragraphs describe the contents for one entry within a category:

```
ICON#=iconfile,scriptfile,infofile,terminal,warning
```

Where:

- `#` = The number of the icon in the KTP. Icons are numbered sequentially, starting at 1.
- `iconfile` = The bitmap icon filename.
- `scriptfile` is the file containing the command to be executed. An example is provided below:

```
<KTP>
COMMAND=(the full pathname of the program to start)
DESCRIBE=(a line description of the program)
```

- `infofile` = The file you create that contains help text for the program. The help file is read as an ASCII file and can be created with any text editor.
- `terminal` = The flag that specifies whether the program needs to be started in its own terminal window. Programs that do not require terminal windows are those that have their own graphical user interface, such as the Keithley Interactive Test Tool (KITT). The valid entries for this input are `Y` for yes and `N` for no.
- `warning` = The flag that specifies whether you should be warned before executing the program. The valid entries for this input are `Y` for yes and `N` for no.

All of the required script files (file type `.src`) must reside in the `$KIDAT/` directory.

When you put all of this together, you get an entry in your `tools.ini` file that looks like this:

```
<KEITHLEY-TOOLS>
ICON1=kitt.ico,kitt.src,kitt.inf,N,N
```

You can modify the programs that are activated from one of the major categories by adding the correct definition for your additional tool. Ensure that the required files are located in the proper directories.

Here is an example of the `tool.tpi` file:

```
<Keithley-Tools>
ICON1=config.ico,config.scr,config.inf,Y,N
ICON2=diags.ico,diags.scr,diags.inf,Y,Y
ICON3=ksu.ico,ksu.inf,N,N
ICON4=download.ico,download.scr,download.inf,Y,Y
ICON5=kitt.ico,kitt.src,kitt.inf,N,N
ICON6=limitx.ico,limitx.scr,limitx.inf,N,N
ICON7=ktpm.ico,ktpm.scr,ktpm.inf,N,N
ICON8=wdu.ico,wdu.scr,wdu.inf,N,N
ICON9=kop.ico,kop.scr,kop.inf,N,N
ICON10=koped.ico,koped.scr,koped.inf,N,N
ICON11=kult.ico,kult.scr,kult.inf,N,N
ICON12=kcat.ico,kcat.scr,kcat.inf,N,N

<Unix-Tools>
ICON1=terminal.ci,terminal.scr,terminal.inf,N,N
ICON2=textedit.ico,textedit.scr,textedit.inbf,N,N
ICON4=calendar.ico,calendar.scr,calendar.inf,N,N
ICON5=clock.ico,clock.scr,clock.inf,N,N
ICON6=mail.ico,mail.scr,mail.inf,N,N

<Unix Administration>
ICON1=xman.ico,xman.scr,xman.inf,N,N

<User Programs>
ICON1=user.ico,user.scr,user.inf,N,N
```

Here is an example of the script file for KITT:

```
<KTP>
COMMAND=kitt
DESCRIBE=StartstheKeithleyInterativeTestTool.
```

System integration

To simplify integrating the Keithley Test Environment (KTE) system into your testing environment, Keithley has developed a series of test libraries that contain commands used during test plan generation, commonly used test procedures, system prober commands, commands used to control and monitor testing, and commands used to handle and store test data. The following paragraphs discuss these different libraries and provide information on test data handling.

Linear Parametric Test Library

The Linear Parametric Test Library (LPTLib) is a high-speed data acquisition and instrument control software system. It is the lowest level of command interface to the systems instrumentation. This library contains commands to program the system instrumentation for parametric testing. Access to this library is available through the Keithley Interactive Test Tool (KITT) and the Keithley User Library Tool (KULT).

See Using function libraries (on page 5-1) for more information about how to use these commands for common applications. For detailed descriptions of the commands, refer to the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Parametric Test Subroutine Library

The S540 Parametric Test Subroutine Library (PARLib) is a collection of standard parametric tests. They can be included in test macros created in the Keithley Interactive Test Tool (KITT) and the Keithley User Library Tool (KULT). These common test procedures have been simplified to avoid repetitious entry of test calls. You only need to fill in the pertinent data for each call. For detailed descriptions of the PARLib commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Keithley Data Files library

The Keithley Data Files (KDF) library is a series of routines to organize and save parametric test data into simple ASCII data files. KDF is discussed further in Data logging (on page 6-174) and in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Keithley User Interface library

The Keithley User Interface (KUI) library contains a series of subroutines that support the creation of user interface dialogs for your test programs. Using KUI, you can control and monitor test activities during the test process. For more information about the KUI subroutines, refer to the "Keithley User Interface Library command reference" in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Data logging

The following paragraphs describe control and handling of the test data.

Keithley Data Files library

The Keithley Data Files (KDF) library is a set of routines to organize and save parametric test data into simple ASCII data files. The following paragraphs discuss the `.kdf` file and how to control the output of this data once it has been stored. For descriptions of the KDF data logging routines, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

KDF uses the Keithley initialization file, `kth.ini`, to define the directory used to store and retrieve data files. By default, all data files are created in the `$KIHOME/db` directory. You can modify this by changing the `kth.ini` file.

All programs that contain KDF library commands must include the `kdf.h` header file, located in the `$KIINCLUDE` directory. This file declares the functions and allows the ANSI-C compiler to validate calling arguments. The proper way to declare a KDF function is:

```
#include "kdf.h"
```

File structure

A Keithley data file (`.kdf`) is stored as a tagged ASCII file. The lot level header information varies in length and is terminated by the `<EOH>` symbol. It is followed by one or more wafer blocks, each of which is terminated with the `<EOW>` symbol. Each wafer block is made up of one or more site blocks, each of which contains the result data for a single site. A site block within a wafer block is terminated by the `<EOS>` symbol. Refer to the example in the following table.

KDF output description file

Data file record	Description of data
`TYP,KDFV1.0`	Identifies the file as a valid V1.0 KDF Data File
`LOT,test1`	Lot ID="test1.kdf"
`PRC,CMOS`	Process="CMOS"
`DEV,TNG-121`	Device="TNG-121"
`TST,Test1`	Test name="Test1"
`TSN,1`	Test Station Number=1
`LMT,limits`	Limits Filename="limits.klf"
`<EOH>`	<End of Header>
`1,,0,0`	Wafer Id = 1
`1,0,0`	Site Id = 1
`1,1.0000e+00`	Tag=1, Value=1.0
`2,3.0000e+00`	Tag=2, Value=3.0
`<EOS>`	<End of sight data>
`<EOW>`	<End of Wafer data>

The lot level header is a variable length structure. Each record in the header section is tagged with the identifiers in the following table.

Optional lot-header parameters

Lot header parameters	KDF tag	Length	Description
id	LOT	51	50 character lot identification (used for lot name)
process	PRC	51	50 character process name
device	DEV	51	50 character device name
testname	TST	256	255 character test name
system	SYS	21	20 character system identification
teststation	TSN	integer	Test station number (1 to 4)
operator	OPR	31	30 character operator name
starttime	STT	21	20 character test start time
stoptime	SPT	21	20 character test completion time
sk1	SK1	31	30 character User Search Key #1
sk2	SK2	21	20 character User Search Key #2
sk3	SK3	11	10 character User Search Key #3
limitcode	LMT	81	80 character limits filename
comment	COM	257	256 character user-definable comment

Keithley data file storage limits

A summary report of the test results is provided by the Keithley Summary Utility (KSU). This report combines the test results from the `.kdf` file and the assigned limits located in the `.klf` file. When combined into a report, you can see which results met or exceeded the specified limits.

Keithley Data File library commands

The following tables contain a brief description of each of the Keithley Data File library (KDF) commands. For detailed descriptions of these commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Data logging routines

Command	Description
PutLot	Logs header information to the DB or FF.
PutWafer	Logs the information in the Wafer Structure to the DB or FF for a specific instance of a lot.
PutSite	Logs the information contained in the Site Structure to the DB or FF for a specific instance of a lot and wafer.
PutParam	Logs the information contained in the Param Structure to the DB or FF for a specific instance of a lot, wafer, and site.
PutParamList	Logs a list of param Structures to the DB or FF for a specific instance of a lot, wafer, and site.
EndLot	Ends the logging of the current lot.
EndWafer	Ends the logging of the current wafer.
EndSite	Ends the logging of the current site.
GetLot	Returns a NULL terminated list of lots, starting with LotStructGot, that match the criteria specified in LotStructWanted.
GetWafer	Returns a NULL terminated list of wafers, starting with WaferStructGot, that match the criteria specified in WafStructWanted for the specific (single) LotStruct.
GetSite	Returns a NULL terminated list of sites, starting with *SiteStructGot, that match the criteria specified in SiteStructWanted for the specific (single) LotStruct and WafStruct.
GetParam	Returns a NULL terminated list of parameters, starting with the *ParamStructGot, that match the criteria specified in ParamStructWanted for the specific (single) LotStruct, WafStruct, and SiteStruct.
GetParamList	Returns a NULL terminated list of parameters to *ParamStruct that are included in the *ParamStructList list.
GetLotData	Returns a tree structure of all the wafers, sites, and parameters in LotWanted.
MatchParam2Limit	Takes the list of parameters and matches them to the corresponding limit codes.
FileExist	Checks for the existence of a file in the current data directory.
LotExist	Checks for the existence of a lot file in the current data directory.
GetStartTime	Returns a time and date string.
DeleteLot	Deletes all lot-associated data for the specified lot structure.
DeleteWafer	Deletes all wafer information for the specified lot and wafer.
DeleteSite	Deletes all site information for the specified lot, wafer, and site.
DeleteParam	Deletes the parameter information for the specified lot, wafer, site, and parameter.
DeleteLimitCode	Deletes entire sets of limits defined by a limit code.
DeleteLimit	Deletes limit records from the DB.

Update comment routines

Command	Description
GetComment	Fetches the comment from the .kdf file for a specific lot occurrence.
PutComment	Overwrites the comment in the .kdf file for a specific lot occurrence.

Update limits routines

Command	Description
GetLimitCode	Fetches a list of limit codes that match the criteria specified in the wanted structure.
GetLimit	Fetches a NULL terminated linked list of limit structures with the specified limit code and limit information.
PutLimit	Writes a list of limits to the DB or FF for the limit code specified.

Structure handling routines

Command	Description
AddNew[STRUCTURE]	Adds new to the list following current.
CreateNew[STRUCTURE]	Allocates the memory for and returns a pointer to the new LIMIT CODE, LOT, WAFER, SITE, or PARAM.
FindFirst[STRUCTURE]	Returns the first LIMIT CODE, LOT, WAFER, SITE, or PARAM that current points to in the list.
FindLast[STRUCTURE]	Returns the last LIMIT CODE, LOT, WAFER, SITE, or PARAM that current points to in the list.
FindNext[STRUCTURE]	Finds the next LIMIT CODE, LOT, WAFER, SITE, or PARAM after the position that current points to in the list.
FindPrev[STRUCTURE]	Finds the previous LIMIT CODE, LOT, WAFER, SITE, or PARAM before the position current points to in the list.
InsertNew[STRUCTURE]	Adds new into the LIMIT CODE LOT, WAFER, SITE, or PARAM list before current.
Remove[STRUCTURE]	Removes the LIMIT CODE, LOT, WAFER, SITE, or PARAM pointed to by current and returns a pointer to the next limit code, lot, wafer, site, or parameter in the list.
LimitExist	Tests for the existence of a limit file in the current Limit File data directory based on the limit code "limitin."

Using limits files

All parametric test results (PTRs) are logged into storage with either the `LogPtr` or `PutParam` commands. Each command logs the result data with an associated result tag.

A limits file associates result tags with result names, units of measurement, and result limits. Use the Keithley Limits File Editor to create new limit files or edit existing ones. Limits files are created with a `.klf` extension, Keithley limit file, and must be located in the same directory as the `.kdf` data files.

Normally there is a single limits file for each of your test programs, but you may decide to create a single master limits file that is shared by all your test programs.

The limits file is only used by the Summary Report Generator. It contains the information required to properly label the individual results and to determine whether the results pass or fail the different validation limits. If the Summary Report Generator is run without a limits file, all validation limits default to ±1E16.

Limits file structure

A limits file is an ASCII file organized into records. Each record consists of multiple attributes that identify and process each measured test result. The following list describes each attribute:

- `ID`, limit record ID tag
- `NAM`, limit record name
- `UNT`, limit record units
- `CAT`, category description
- `RPT`, report option: 1 = use in Summary Report, 0 = ignore
- `CRT`, critical level
- `TAR`, target value of parameter
- `AF`, abort flag setting
- `AL`, limit abort flag is to be compared to
- `VAL`, low and high VALID limits
- `SPC`, low and high SPECIFICATION limits
- `CNT`, low and high CONTROL limits
- `ENG`, low and high ENGINEERING limits
- `ena`, enabled flag (only for adaptive test): 1 = test, 0 = ignore
- `cla`, class data field
- `usr1`, user data field
- `usr2`, user data field

- usr3, user data field
- <EOL>, marks end of limit record

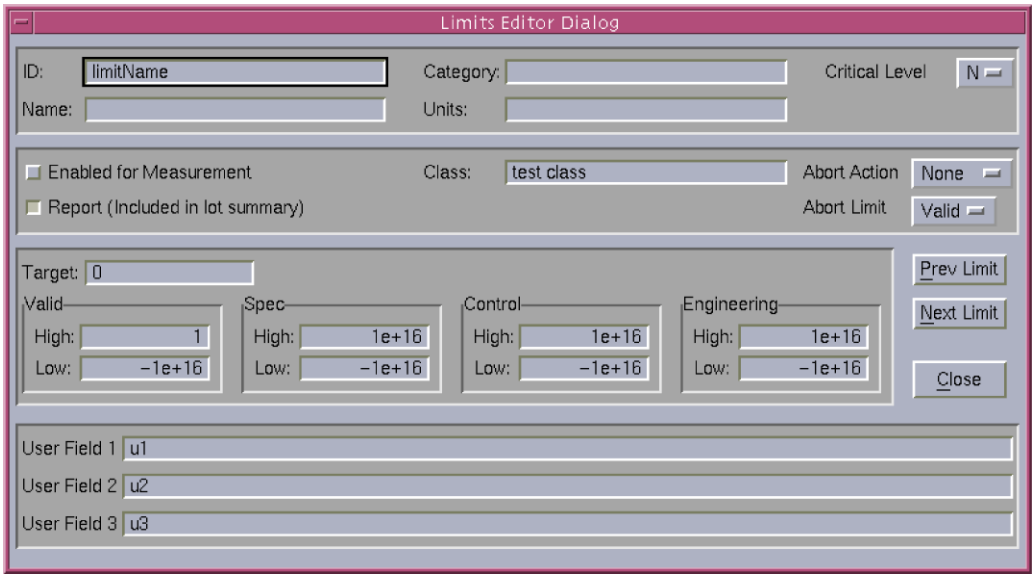
When a limits file is loaded into memory, the cla, usr1, usr2, and usr3 fields contain a NULL pointer, not an empty string, if the data is not defined in the limits file. The NAM, UNT, and CAT fields contain an empty string if the data is not defined in the limits file.

These attributes are repeated for each result ID in the limits file. The following example limits file contains the file header and two separate result ID records:

```
Version,1.0
File,/home/kiSys/db/tutor.klf
Date,07/21/2018
Comment,KLF
<EOH>
ID,moda_n13x1_vtati
NAM,moda_n13x1_vtati
UNT,volts
CAT,test
RPT,1
CRT,1
TAR,1.32
AF,N
AL,VAL
VAL,-1.0000e+16, 1.0000
SPC,-1.0000e+16, 1.0000e+16
CNT,-1.0000e+16, 1.0000e+16
ENG,-1.0000e+16, 1.0000e+16
ENA,1
CLA,main
USR1,user 1
USR2,user 2
USR3,user 3
<EOL>
ID,i_res
NAM,ires test
UNT,ohms
CAT,test
RPT,1
CRT,0
TAR,1.000000
AF,SS
AL,SPC
VAL,-1.0000e+20, 1.0000e+21
SPC,-1.0000e+16, 1.0000e+16
CNT,-1.0000e+16, 1.0000e+16
ENG,-1.0000e+16, 1.0000e+16
ENA,1
CLA,main
USR1,user 1
USR2,user 2
USR3,user 3
<EOL>
```

Each result ID record within a limits file can be viewed individually. By selecting **Dialog** from the **View** menu, you can view and modify the currently selected limit in the Limits Editor Dialog window, shown in the following figure.

Figure 98: Limits editor dialog window



Programming examples

The following topics contain programming examples.

Logging data using KDF

The following sample program demonstrates how to log data to a `.kdf` file. It loops through two simulated wafers, each with three sites, and logs two parameters per site. The programs use both of the available data logging interfaces to create the same output file. The final output files are named and displayed after the source code listing.

```
#include <stdio.h>
#include <stdlib.h>
#include "kdf.h"
void main(void)
{
LOT*testlot;
WAFER*testwafer;
SITE*testsite;
PARAM*testparam;
int waferloop,siteloop;
int status;
int total_wafers=2;
int sites_per_wafer=3;
floatvalue;
charwafer_id[4];
charsite_id[4];
```

```
/*************** S T A R T  O F  C O D E *****************/
/* Initialize kdf data structures */
testlot = CreateNewLot ();
testwafer = CreateNewWafer();
testsite = CreateNewSite ();
testparam = CreateNewParam();

/* Initialize some of the lot header items */
strcpy (testlot->id, "test1");       /* lot name */
strcpy (testlot->process, "CMOS");   /* process name */
strcpy (testlot->device, "TNG-121"); /* device name */
strcpy (testlot->testname, "Test1"); /* test name */
strcpy (testlot->limitcode, "limits");     /* limits filename */
testlot->teststation=1; /* test station 1 */

/* Actually create the lot file */
status = PutLot(testlot, CREATELOT);
if (status < 0) exit(status);
printf("\n\nStart Test #1\n");

/* Start Testing: Loop through all wafers & sites */
tstsel(1);
for (waferloop = 1;waferloop <= total_wafers;waferloop++)
{
   sprintf(testwafer->id,"%i",waferloop);
   status = PutWafer(testlot,testwafer);
   if (status < 0) exit(status);
   printf("Logging Wafer %i\n", waferloop);

   for (siteloop = 1; siteloop <= sites_per_wafer; siteloop++)
   {
          sprintf(testsite->id,"%i",siteloop);
          status = PutSite(testlot,testwafer,testsite);
          if (status < 0) exit(status);
          printf("\tLogging Site %i\n", siteloop);

          /* calculate the beta*/
          strcpy(testparam->id, "1");
          testparam->value = beta1(1,4,7,-1, 0.5e-3, 2.0, 'N');
          status = PutParam(testlot,testwafer,testsite, testparam);
          if (status < 0) exit(status);

          /* test the capacitance when the bias voltage is 1.0 */
          strcpy(testparam->id, "2");
          testparam->value = cap(3,5,7,1.0);
          status = PutParam(testlot,testwafer,testsite, testparam);
          if (status < 0) exit(status);

          /* Site complete */
          EndSite();

          /* Insert prober code here to move to next site */
}

/* Wafer complete */
EndWafer();
```

```
   /* Insert prober code here to move to next site */
}

/* Lot complete. Close data file */
EndLot();

/* Initialize kdf data structures */

status = LogLot("test2", 0, "CMOS", "TNG-121", "Test1", "", "",
"Limits", "", CREATELOT);
if (status <0) printf("Error in LogLot, status %i\n",status);

/* Start Testing: Loop through all wafers & sites */
tstsel(1);
printf("\n\nStart Test #2\n");
for (waferloop = 1;waferloop <= total_wafers;waferloop++)
{

   sprintf(wafer_id,"%i",waferloop);
   status = LogWaf(wafer_id,0);
   if (status <0) printf("Error in LogWaf, status %i\n",status);
   printf("Logging Wafer %i\n", waferloop);

   for (siteloop = 1; siteloop <= sites_per_wafer; siteloop++)
   {
          sprintf(site_id,"%i",siteloop);
          status = LogSit(site_id, 0);
          if (status <0) printf("Error in LogSit, status %i\n",status);
          printf("\tLogging Site %i\n", siteloop);

          /* calculate the beta*/
          value = beta1(1, 4, 7, -1, 0.5e-3, 2.0, 'N');
          status = LogPtr(1, value);
          if (status <0) printf("Error in LogPtr, status %i\n",status);
          /* test the capacitance when the bias voltage is 1.0 */
          value = cap(3,5,7,1.0);
          status = LogPtr(2, value);
          if (status <0) printf("Error in LogPtr, status %i\n",status);

          /* Site complete */
          EndSite();

          /* Insert prober code here to move to next site */
   }

   /* Wafer complete */
   EndWafer();

   /* Insert prober code here to move to next site */
}

/* Lot complete. Close data file */
EndLot();
}
```

The two data files created by this example are shown in the following table.

PUTxxx and LotLog/LogWat/LogSit data files

File created by Putxxx Interface	File created by LogLot/LogWaf/ LogSit Interface	File created by Putxxx Interface	File created by LogLot/LogWaf/ LogSit Interface
TYP,KDFV1.0 LOT,test1 PRC,CMOS DEV,TNG-121 TST,Test1 TSN,1 LMT,limits <EOH> 1,,0,0 1,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 2,0,0 1,1.0000e+00 2,3.0000e+00 <EOS>	TYP,KDF V1.0 LOT,test2 PRC,CMOS DEV,TNG-121 TST,Test1 TSN,1 LMT,limits <EOH> 1,,0,0 1,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 2,0,0 1,1.0000e+00 2,3.0000e+00 <EOS>	3,0,0 1,1.0000E+00 2,3.0000E+00 <EOS> <EOW> 2,,0,0 1,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 2,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 3,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> <EOW>	3,0,0 1,1.0000E+00 2,3.0000E+00 <EOS> <EOW> 2,,0,0 1,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 2,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> 3,0,0 1,1.0000e+00 2,3.0000e+00 <EOS> <EOW>

The resulting summary report is shown below:

```
   Lot Summary Report KDF V1.0
Lot : test1       Operator      :
Process    : CMOS Starttime     :
Device     :TNG-121      System :
Testname   : Test1       Teststation   : 1 Limits File: limits
#Wafers    :      2
#Sites     :      6
```

Name units	SpecL	SpecH	Mean	Min	Max	SDEV	%SDEV	CNT	%Vld	%Spec
1	-1.00e+15	1.00e+15	1.00e+00	1.00e+00	0.00e+00	0.00e+00	0.0	6	100.0	100.0
2	-1.00e+15	1.00e+15	3.00e+00	3.00e+00	3.00e+00	0.00e+00	0.0	6	100.0	100.0

Data retrieval using KDF

The following sample program demonstrates how to read data from a `.kdf` file. The program attempts to read all of the data from `test1.kdf` (created in the previous example) and display the results on the screen.

```
#include <stdio.h>
#include <stdlib.h>
#include "kdferr.h"
#include "kdf.h"

void main(void)
{
LOT *searchlot,*gotlot;
WAFER     *searchwafer,*gotwafer;
SITE      *searchsite,*gotsite;
PARAM     *searchparam,*gotparam;

int waferloop,siteloop;
int status;

/************ START DATA RETRIEVAL *****************/
searchlot = CreateNewLot();
gotlot = CreateNewLot();

/* Retrieve all the data from lot test1.kdf */
strcpy(searchlot->id, "test1");
GetLot(searchlot, gotlot);

/* Get all the wafers in the lot */
searchwafer = CreateNewWafer();
gotwafer = CreateNewWafer();
strcpy(searchwafer->id, "*");
GetWafer(gotlot, searchwafer, gotwafer);

while (gotwafer != NULL)
{
   printf("\tWafer %s\n", gotwafer->id);
   searchsite = CreateNewSite();
   gotsite = CreateNewSite();
   /* Get all the sites in this wafer */
   strcpy(searchsite->id, "*");
   GetSite(gotlot, gotwafer, searchsite, gotsite);

   while (gotsite != NULL)
   {
          printf("\t\tSite %s\n", gotsite->id);
          searchparam = CreateNewParam();
          gotparam = CreateNewParam();
          /* Get all the parameters in this site */
          strcpy(searchparam->id, "*");
          GetParam(gotlot, gotwafer, gotsite, searchparam, gotparam);
          while (gotparam != NULL)
          {
                 printf("\t\t\tId, Value = %s, %e\n", gotparam->id, gotparam->value);
```

```
                gotparam = FindNextParam(gotparam);
          }

          gotsite = FindNextSite(gotsite);
    }
    gotwafer = FindNextWafer(gotwafer);
}
/************* CLEANING UP MEMORY *****************/

RemoveParam(searchparam);
RemoveSite (searchsite);
RemoveWafer(searchwafer);
RemoveLot (searchlot);
RemoveLot (gotlot);

while (gotwafer != NULL)
gotwafer = RemoveWafer(gotwafer);

while (gotsite != NULL)
gotsite = RemoveSite(gotsite);

while (gotparam != NULL)
gotparam = RemoveParam(gotparam);
}
```

The resulting output is:

```
Wafer 1
   Site 1
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
   Site 2
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
   Site 3
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
Wafer 2
   Site 1
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
   Site 2
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
   Site 3
          Id, Value = 1, 1.000000e+00
          Id, Value = 2, 3.000000e+00
```

Keithley Data File user tag data

The Keithley Data File (KDF) library supports user-defined data. This information is contained within the `.kdf` file with the following format:

```
<TAG>"tagName",tag value string
```

- The `<TAG>` field is required.
- The `"tagName"` field can contain any characters except the double-quote character. The max length of the `tagName` is `PARAM_ID_LENGTH` (128) characters.
- The comma is required and separates the `tagName` field from the tag value string field.
- The `"tag value string"` has a max length of 512 characters and cannot contain the new-line character.

The following routines have been added to the KDF library:

- `PutTag`
- `GetTag`
- `PrintTagList`
- `ClearTagList`

The application programming interface (API) descriptions are as follows:

```
int PutTag(char *tagName, char *valueString) ;
```

Return values:

- `0` = Success
- `<0` = Error (error codes can be found in the kdferr.h header file)
- `BAD_CALL_ORDER (-14)` = Called before `PutLot()` called
- `ERR_WRITE_FILE (-21)` = Error from fprintf call to write data
- `ERR_OPEN_FILE (-20)` = Error from fopen call
- `ERR_CLOSE_FILE (-22)` = Error from fclose call
- `ILLEGAL_TAG (-25)` = Illegal character in tag name
- `TAG_STR_LEN_ERR (-26)` = Tag string length too long
- `TAG_NAM_LEN_ERR (-27)` = Tag name too long

PutTag will write a user tag into the KDF file using tagName and valueString.

- tagName can contain any printable ASCII character except the double-quote character. Maximum length is PARAM_ID_LENGTH, or 128 characters. tagName must be null-terminated.
- valueString has a maximum length of 512 characters and can contain any printable ASCII. The valueString cannot contain the new-line character. valueString must be null-terminated.

Example:

```
status = PutTag( "myWaferTag", "wafer tested using Base algorithm" ) ;
```

The above code will cause the following data to be written into the .kdf file at the current location:

```
<TAG>"myWaferTag",wafer tested using Base algorithm
int GetTag( LOT *wantedLot,
            WAFER *wantedWafer,
            SITE *wantedSite,
            char *tagName,
            tagList **got ) ;
            typedef struct _tagList
                        {
                  char *tagName ;
                  char *tagString ;
                  _tagList *next ;
            } tagList ;
```

GetTag returns a null-terminated list of tag data whose head is the got parameter. This list is malloced and needs to be freed by the calling routine.

- wantedLot is the specific lot in which the tagName should be found.
- wantedWafer is the specific wafer in which to start searching for tags. If wantedWafer is NULL, all tags for all wafers will be returned in the list. If wantedWafer is a specific wafer, just tags associated with the wafer will be returned. Wafer tags are tags located between the Wafer header line and the <EOW> marker.
- wantedSite is the specific site in which to start searching for tags. If wantedSite is NULL, all tags for all sites on the wafer will be returned in the list. If wantedSite is a specific site, just tags associated with the site will be returned. Site tags are tags located between the site header line and the <EOS> marker.
- tagName is the name of the tag. If tagName is NULL, all tags will be returned in the list or just the tags named tagName will be returned. Wildcard characters are supported in the tagName field.

```
void PrintTagList( tagList *head ) ;
```

PrintTagList is a debug routine that can be used to display the list of tags pointed to by the head parameter. The tags are written to stdout.

```
void ClearTagList( tagList **head ) ;
```

`ClearTagList` is a routine that will free the `tagList` created by a call to `GetTag`.

Example

The following code sequence will return a list of all user tags for all wafers and sites in the specified lot file. The list is pointed to by the `gotTagList` variable. Please note that the tag list is malloced by the `GetTag` routine and must be freed by the user. The `ClearTagList()` routine can be used for this operation.

```
tagList *gotTagList = NULL ;

GetTag( lot, NULL, NULL, NULL, &gotTagList ) ;
PrintTagList( gotTagList ) ;
ClearTagList( &gotTagList ) ; /* free list created by GetTag */
```

Example of use within a cassette plan

You need to add additional data for the lot and wafer. Using the `PutTag` routine at `UAP_WAFER_PREPARE` will allow you to add information after the lot header. `UAP_WAFER_BEGIN` can be used to add information after the wafer header section of the `.kdf` file.

If the following commands are called up at `UAP_WAFER_PREPARE`:

```
PutTag( "LotType", "CMOS" ) ;
PutTag( "LotPlant", "Cleveland" ) ;
```

And the following is called at `UAP_WAFER_BEGIN`:

```
PutTag( "WaferPlanUsed", "WPFbase" ) ;
```

The resulting `.kdf` file will be:

```
TYP,KDF V1.1
LOT,lotName
TST,ktxe sample
SYS,S540q4000
TSN,1
OPR,williamson
STT,2-Nov-2018 13:23
WDF,sample.wdf
<EOH>
<TAG>"LotType",CMOS
<TAG>"LotPlant",Cleveland
Wafer_01,,1,1
<TAG>"WaferPlanUsed",WPFbase
Site_1,1,1
Param1, 5.0000e+00
<EOS>
<EOW>
```

NOTE

In the example above, the system is identified as `S540`; if you have a different system, this number will be different.

User access points

The following is a list of user access points (UAP) within the Keithley Test Execution Engine (KTXE) and when the Keithley Data Files (KDF) logging routines are executed. The `PutTag` routine can be called any time after the `PutLot()` routine is called.

```
UAP_PROG_ARGS
UAP_CASSETTE_LOAD
UAP_LOT_INFO
UAP_PROBER_INIT
UAP_WAFER_MISMATCH
UAP_ACCESS_WDF_INFO
UAP_POST_PROBER_INIT
UAP_WRITE_LOT_INFO
   PutLot()
UAP_POST_LOT_INFO
UAP_ALIGN_ERROR
UAP_POST_INITIAL_WAFER_LOAD

/* Start of Wafer Loop */
UAP_WAFER_PREPARE
UAP_VALIDATE_OCR
   PutWafer()
UAP_WAFER_BEGIN
UAP_SITE_CHANGE
   EndSite()
   PutSite()
UAP_SUBSITE_CHANGE
UAP_TEST_BEGIN
UAP_TEST_END
   PutParam()
UAP_TEST_DATA_LOG
UAP_HANDLE_ABORT
UAP_SUBSITE_END
UAP_SITE_END

   EndSite()
UAP_WAFER_END
   EndWafer()

UAP_ALIGN_ERROR

/* End of Wafer Loop */

   EndLot()
UAP_LOT_END
UAP_ENGINE_EXIT
UAP_ABORT_EXIT_HDLR
```

Structure definitions

For structure definitions of `LOT`, `WAFER`, `SUBSITE`, `SITE PARAM`, `LIMITCODE`, and `LIMIT`, refer to the `include` file in `$KIINCLUDE/kdf.h`.

Sample programs

The following topics contain sample programs.

Logging one PARAM at a time, data retrieval through Get routines

```
#include <stdio.h>
#include <stdlib.h>
#include "kdferr.h"
#include "kdf.h"

void main(void)
{
LOT *testlot,     *gotlot;
WAFER     *testwafer,   *gotwafer;
SITE      *testsite,    *gotsite;
PARAM     *testparam,   *gotparam;

int waferloop,siteloop;

int status;

testlot   = CreateNewLot  ();
testwafer = CreateNewWafer();
testsite  = CreateNewSite ();
testparam = CreateNewParam();

strcpy (testlot->id, "test1");
strcpy (testlot->testname, "Voltage 1");
GetStartTime(testlot->starttime);

/* Start the logging of the new lot "test1". The lot has 3 wafers and 10
sites to be logged.
*/
status = PutLot(testlot, CREATELOT);
if (status < 0)
return(status);

for (waferloop = 1;waferloop <= 3;waferloop++)
{
sprintf(testwafer->id,"%i",waferloop);
status = PutWafer(testlot,testwafer);
if (status < 0)
return(status);
```

```
for (siteloop = 1; siteloop <= 10; siteloop++)
{
sprintf(testsite->id,"%i",siteloop);
status = PutSite(testlot,testwafer,testsite);
if (status < 0)
return(status);

strcpy(testparam->id, "beta1");
/* calculate the beta*/
testparam->value = beta1(1, 4, 7, -1, 0.5e-3, 2.0, 'N');
status = PutParam(testlot,testwafer,testsite,testparam);
if (status < 0)
return(status);

strcpy(testparam->id, "cap 1.0");
/* test the capacitance when the bias voltage is 1.0 (Parlib routine)*/
testparam->value = cap(3,5,7,1.0);
status = PutParam(testlot,testwafer,testsite,testparam);
if (status < 0)
return(status);

EndSite();
/* Prompt user to move to the next site */
printf("Please move the prober to the next site and hit a key\n");
getchar();
}

EndWafer();
/*Prompt user to move to the next wafer */
printf("Please move the prober to the next wafer and hit a key\n")
getchar();
}
EndLot();
GetStartTime(testlot->stoptime);

/* Append the stoptime into the header. (Does not change any of the data that
was already logged.)
*/
status = PutLot(testlot,APPENDLOT);
if (status < 0)
return(status);

/************          END OF DATA LOGGING  *****************/
/************          START DATA RETRIEVAL *****************/

gotlot = CreateNewLot();

strcpy(testlot->id, "test1");
/* Retrieve all the data that was just logged */
GetLot(testlot, gotlot);

gotwafer = CreateNewWafer();

/* Get all the wafers in the lot */
strcpy(testwafer->id, "*");
GetWafer(gotlot, testwafer, gotwafer);
```

```
while (gotwafer != NULL)
{
gotsite = CreateNewSite();

/* Get all the sites in this wafer */
strcpy(testsite->id, "*");
GetSite(gotlot, gotwafer, testsite, gotsite);

while (gotsite != NULL)
{
gotparam = CreateNewParam();

/* Get all the parameters in this site */
strcpy(testparam->id, "*");
GetParam(gotlot, gotwafer, gotsite, testparam, gotparam);

/* If the first parameter result is greater than 1, remove that site from the lot */
if (gotparam->value > 1)
DeleteSite(gotlot,gotwafer,gotsite)

gotsite = FindNextSite(gotsite);
}
gotwafer = FindNextWafer(gotwafer);
}

/************          CLEANING UP MEMORY   *****************/
RemoveParam(testparam);
RemoveSite (testsite);
RemoveWafer(testwafer);
RemoveLot  (testlot);

RemoveLot  (gotlot);
while (gotwafer)
gotwafer = RemoveWafer(gotwafer);
while (gotsite)
gotsite = RemoveSite(gotsite);
while (gotparam)
gotparam = RemoveParam(gotparam);
}
```

Logging a linked list of PARAMs, data retrieval using GetLotData

```
#include <stdio.h>
#include <stdlib.h>
#include "kdferr.h"
#include "kdf.h"

void main(void)
{
LOT *testlot,    *gotlot;
WAFER     *testwafer,   *gotwafer;
SITE      *testsite,    *gotsite;
PARAM     *testparam,   *gotparam;
PARAM *new;
int waferloop,siteloop;
int status;

testlot   = CreateNewLot  ();
testwafer = CreateNewWafer();
testsite  = CreateNewSite ();

strcpy (testlot->id, "test1");
strcpy (testlot->testname, "Voltage 1");
GetStartTime(testlot->starttime);

/* Start the logging of the new lot "test1". The lot has 3 wafers and 10
sites to be logged.
*/
status = PutLot(testlot, CREATELOT);
if (status < 0)
return(status);

for (waferloop = 1;waferloop <= 3;waferloop++)
{
sprintf(testwafer->id,"%i",waferloop);
status = PutWafer(testlot,testwafer);
if (status < 0)
return(status);

for (siteloop = 1; siteloop <= 10; siteloop++)
{
sprintf(testsite->id,"%i",siteloop);
status = PutSite(testlot,testwafer,testsite);
if (status < 0)
return(status);

testparam=CreateNewParam();
strcpy(testparam->id, "Volts 1e-2");
/* voltagetest is an example test routine that would return a voltage */
testparam->value = voltagetest(1e-2);

new=CreateNewParam();

strcpy(new->id, "Volts 1e-1");
   /* voltagetest is an example test routine that would return a voltage */
```

```
testparam->value = voltagetest(1e-1);
/* Add current into the list following testparam */
AddNewParam(testparam,new);
testparam = FindNextParam(testparam);
new = CreateNewParam();

strcpy(new->id, "Volts 1");
/* voltagetest is an example test routine that would return a voltage */
testparam->value = voltagetest(1);
/* Add current into the list following testparam */
AddNewParam(testparam,new);

/* Go to the first param in the list to get ready for PutParamList */
testparam=FindFirstParam(testparam);
PutParamList(testlot,testwafer,testsite,testparam);

EndSite();
/* Some routine to get the next site ready for testing */
move_next_site();
}

EndWafer();
/* Some routine to get the next wafer ready for testing */
move_next_wafer();
}
EndLot();
GetStartTime(testlot->stoptime);

/* Append the stoptime into the header. (Does not change any of the data that was already
   logged.)
*/
status = PutLot(testlot,APPENDLOT);
   /************       END OF DATA LOGGING  *****************/
/************          CLEANING UP MEMORY   *****************/
while(testparam != NULL)
RemoveParam(testparam);
RemoveSite (testsite);
RemoveWafer(testwafer);
RemoveLot  (testlot);
/************          START DATA RETRIEVAL *****************/

gotlot = CreateNewLot();

strcpy(gotlot->id, "test1");
/* Retrieve all the data that was just logged */
GetLotData(gotlot);

gotwafer = gotlot->wafers;

while (gotwafer != NULL)
{
gotsite = gotwafer->sites;

while (gotsite != NULL)
{
gotparam = gotsite->params;
```

```
/* If the result was greater than 1, remove that site from the lot */
if (gotparam->value > 1)
DeleteSite(gotlot,gotwafer,gotsite)

gotsite = FindNextSite(gotsite);
}
gotwafer = FindNextWafer(gotwafer);
}
}
```

Keithley User Interface Library

The Keithley User Interface (KUI) Library contains commands that provide the program developer with a basic set of user interfaces for operator data entry and program status monitoring program specifically for test programs.

The location of the library and files depends on the specific platform and product. Refer to the specific platform's manual for location of the library.

An `include` file named `kui_proto.h` is provided for the library. The following line should be placed in every test program or file that will be linked with the KUI library:

```
#include "kui_proto.h"
```

Starting with Keithley Test Environment (KTE) software version 5.1.0 and later, the following KUI window routines are displayed on the Dialog tab on the Keithley Integrated Display Service (KIDS) interface. If you would rather have the old-style windows display, set the `KI_KUI_CLASSIC` environment variable to `1`.

If the KIDS interface is not running or connections are not being accepted by KIDS, KUI defaults to the Classic windowing mode.

Note that the WfrIdDlg window is not supported by the KIDS interface. The Classic windows (Keithley Operator Interface (KOP)) is used for WfrIdDlg routines.

User interface constants

Constants corresponding to values returned by the dialogs indicate how the dialog was exited. The returned value is then used to determine further program execution.

`DLG_ABORT`

`DLG_EXIT`

`DLG_NO`

`DLG_OK`

`DLG_SKIP`

`DLG_YES`

Constants that can be used to determine the look and feel of the user interface dialogs. They are intended for use with the `InitUI` command.

`DLG_LOOK_MOTIF`

`DLG_LOOK_MSW`

`DLG_LOOK_OPENLOOK`

`DLG_LOOK_PM`

`DLG_LOOK_PM2`

Constants are provided for those dialogs which allow selecting which of the dialog's fields can be edited by the user by passing a field edit enable array as a part of a dialog's command call.

`FIELD_ENABLED`

`FIELD_DISABLED`

Constants intended for use with the pause, continue, and abort flag (`pca_flag`) used by the Status dialog. Refer to the `StatusDlg` command in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

`KI_ABORT`

`KI_CONTINUE`

`KI_PAUSE`

Constant that specifies the number of fields within the Lot Information Dialog (`LotDlg`). Used to declare the `lot_dlg_flds` array with the proper dimension.

`NUM_LOT_FIELDS`

Constant that indicates the highest value allowed for dialog look selection. Refer to the `GetProgramArg` GUI look command-line argument switch and the `InitUINew` command in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

`MAX_DLG_LOOK`

Enums corresponding to fields in the Lot Information Dialog (`LotDlg`). Intended for use as indexes into the `lot_dlg_flds` array in order to select `FIELD_ENABLED` or `FIELD_DISABLED`. Refer to LotDlg in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

`EXIT_LOT_DLG`

`OPERATOR`

`LOT_ID`

`PROCESS`

DEVICE

TEST_NAME

SYSTEM_ID

TEST_STATION

SEARCH_KEY1

SEARCH_KEY2

SEARCH_KEY3

LIMIT_FILE

LOT_COMMENT

User interface library variables

lot_dlg_fields[]

Array that assigns an element to each field in the Lot Information Dialog (LotDlg). The value for the element determines whether the field can be edited by the operator when the lot information dialog is displayed. It is declared within the LotDlg source file and defined as an external int array in the header file to make it accessible to the user program. Refer to LotDlg.

pca_flag

Variable that flags the execution state of the test plan as controlled by the Status Dialog. There is a routine that can be used to query the state of the pca_flag.

```
int Get_pca_flag;
```

If the Keithley Test Execution Engine (KTXE) is running with the Keithley User Interface (KUI) disabled, using the Get_pca_flag() routine will allow your user access point (UAP) code to detect that a tester fatal event has occurred and can force the engine to exit. The KTXE UpdateStatusAbort call within the execution engine checks this flag automatically. Your UAP code can also check the flag to detect state changes earlier.

User interface library commands

The following table provides a brief description of the Keithley user interface (KUI) user library commands.

For detailed descriptions of the KUI user library commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`GetProgramArgs`	Gets program command-line arguments.
`InitUINew`	Initializes the user interface library.
`InputMsgDlg`	Displays the input message dialog.
`LotDlg`	Displays a modal dialog window to collect lot information from the operator.
`OkCancelAbortMsgDlg`	Prompts the operator to acknowledge or cancel the passed abort message string.
`OkCancelMsgDlg`	Prompts the operator to acknowledge or cancel the passed message string.
`OkMsgDlg`	Prompts the operator to acknowledge the passed message string.
`QuitUI`	Cleans up user interface before program exit.
`ScrollMsgDlg`	Sets a label at the top of a scrollable dialog box.
`ScrollMsgDlgClr`	Clears all message from the scrolling message dialog box.
`ScrollMsgDlgMsg`	Posts a message to the scrolling message dialog box.
`StatusDlg`	Creates a persistent modeless dialog box that remains visible until `QuitUI` releases it.
`UpdateModelessDlgs`	Updates the scrolled message dialog box when contents change.
`UpdateStatusDlg`	Updates the displayed status dialog box fields when changed in the test program.
`VarMsgDlg`	Creates a window with scrolling text and multiple push buttons.
`WfrIdsDlg`	Displays a modal dialog window to collect information from the operator for multiple wafers to be tested by the test program.
`WfrIdDlg`	Displays a modal dialog window to collect information from the operator for single wafers to be tested by the test program.
`YesNoAbortMsgDlg`	Displays a modal dialog window containing the passed message string and requires operator acknowledgement by selecting YES, NO, or ABORT.
`YesNoCancelMsgDlg`	Displays a modal dialog window containing the passed message string and requires operator acknowledgement by selecting YES, NO, or CANCEL.
`ContSkipAbortDlg`	Displays a message in a dialog box with the Continue, Skip, and Abort selections.
`LBoxDlg`	Opens a window containing a selection list of one or more items.

KTE KUI localization

The KI_LOCALIZE_CFG environment variable is used to define an alternate localization file. Simply set KI_LOCALIZE_CFG to a file of the form below, and localization will automatically take place within The Keithley User Interface (KUI). Only the elements that you want to modify need to be included in this file.

Note that all defaults are included in this example. Format for the lines of the file requires:

```
name,"value"
```

Note that other comments may come after the second unescaped quotation mark on each line, and that blank lines and those starting with a pound sign (#) will be ignored.

NOTE

This feature is only used for the KUI_CLASSIC graphical user interface (GUI) windows. These values will have no effect in the KTE Integrated Display Service (KIDS) GUI display. The guiLabels.xml and ktxe_error_msgs.xml files are used to define the label values and messages for the KIDS display. This path name for these files is determined by the pathToXMLfiles setting in the kth.ini file.

Example:

```
#Keithley Localization File
Version,1.0
File,/opt/ki/dat/example.loc
Date,
Id,
Comment,
<EOH>
KI_LOTINFO_LBL,"Lot Information - " # Lot Info Title
KI_OPR_LBL,"Operator"
KI_LOT_LBL,"Lot Id"
KI_PROCESS_LBL,"Process"
KI_DEVICE_LBL,"Device"
KI_TESTNAME_LBL,"Test Name"
KI_LIMIT_LBL,"Limit Id"
KI_SYS_LBL,"System Id"
KI_TESTSTN_LBL,"Test Station"
KI_SK1_LBL,"Search 1"
KI_SK2_LBL,"Search 2"
KI_SK3_LBL,"Search 3"
KI_COMMENT_LBL,"Comment"
KI_STATUS_LBL,"Status - " # Status Dlg Title
KI_CPFNAME_LBL,"Cass. Plan"
KI_WPFNAME_LBL,"Wafer Plan"
KI_PCFNAME_LBL,"Probe Card"
KI_GDFNAME_LBL,"Global Data"
KI_WDFNAME_LBL,"Wafer Desc"
KI_TIME_LBL,"Total Time"
```

```
KI_WAFID_LBL,"Wafer Id"
KI_CURWAF_LBL,"Wafer"
KI_OF_LBL,"Of"
KI_SLOT_LBL,"Slot"
KI_SPLIT_LBL,"Split"
KI_CASS_LBL,"Cassette"
KI_X_LBL,"X"
KI_Y_LBL,"Y"
KI_COL_LBL,"X"
KI_SSID_LBL,"SubSite"
KI_ROW_LBL,"Y"
KI_SID_LBL,"Site"
KI_OF_LBL,"Of"
KI_WFRIDSDLG_LBL,"Wafer Information - " # Wafer Ids Dlg title
KI_WFRIDDLG_LBL,"Wafer Information - " # Wafer ID Dlg Title
KI_CBCASS_LBL,"Cassette"
KI_CBSLOT_LBL,"Slot"
KI_WFRIDCOL_LBL,"ID"
KI_WFRSPLITCOL_LBL,"SPLIT"
KI_OKBUT_LBL,"OK"
KI_ABORTBUT_LBL,"ABORT"
KI_HELPBUT_LBL,"Help"
KI_YESBUT_LBL,"Yes"
KI_NOBUT_LBL,"No"
KI_SKIPBUT_LBL,"Skip"
KI_CANCELBUT_LBL,"Cancel"
KI_CONTTBUT_LBL,"Continue"

I_ERROR_LBL,"Error Log for "
KI_EVENT_LBL,"Event Log for "

#
#-------------------------------------------------------------------------
# The following are text for System Message Dialog windows
#-------------------------------------------------------------------------
#
KI_SUSPEND_ABORT_MSG,"Abort KTXE?"
KI_ABORT_EXE_MSG,"Are you sure you want to ABORT execution?\n"
KI_SUSPEND_UNLOAD_MSG,"You MUST unload all wafers MANUALLY!"

KI_LOT_EXISTS_USE_MSG,"Lot Data already exists and may be in use!\n\nDo you want to
   append data to the lot?\n\nPress YES to append to the existing lot data.\nPress NO
   to DELETE the old lot data\nand create a NEW lot data file.\nPress ABORT to stop
   execution."

KI_LOT_EXISTS_MSG,"Lot Data already exists!\n\nDo you want to append data to the
   lot?\n\nPress YES to append to the existing lot data.\nPress NO to DELETE the old
   lot data\nand create a NEW lot data file.\nPress ABORT to stop execution."

KI_OK_CANCEL_MSG,"Press OK to Continue\nCancel to ABORT"
KI_LOAD_CASS_MSG,"Load Wafer Cassette(s).\n\nWhen ready press OK to continue or CANCEL
   to ABORT test program"

KI_LOAD_WAFER_MSG,"Load/Unload Wafer from chuck"
KI_FRONT_LOAD_MSG,"Manually Load/Profile/Align Wafer.\n\nMove chuck to target die"
KI_FRONT_UNLOAD_MSG,"Manually UnLoad Wafer"
```

```
KI_NO_MAP_MSG,"Could not map any cassettes."

#
#--------------------------------------------------------------------------
# The following values have sprintf arguments embedded within. The arguments
# MUST EXIST in any changes that you make
#--------------------------------------------------------------------------
#
KI_PROBER_ERROR_MSG,"PROBER ERROR %i has occurred.\nPlease clear the error.\nPress
   'Continue' for current wafer\nPress 'Skip' to skip current wafer\nPress 'Abort' to
   ABORT the test program."

KI_KDF_ERROR_MSG,"KTXE ERROR, %s returned status = %d\nYou can attempt to correct the
   problem\nand retry or abort the test program\n\nDo you wish to retry %s?\n\nPress
   OK to RETRY the operation\nPress CANCEL to ABORT\n"

KI_RESUME_MSG,"Resume lot id: %s ??"

KI_CASS_NOT_LOADED_MSG,"Cassette %d Not loaded"

KI_CASS_UNMAP_MSG,"Unmapped wafers found in cassette %d. Please wait for mapping to
   complete"

<EOLOC>
```

KTE file formats

The following topics contain examples of the data files that are created by the different tools in the Keithley Test Environment (KTE) software. The extensions for each of the data files are:

- `.wdf`: The Wafer Description File created by the Wafer Description Utility (WDU).
- `.tsf`: The Test Structure File created by the Test Structure Editor (TSE).
- `.ktm`: The Keithley Test Macro created by the Keithley Interactive Test Tool (KITT).
- `.pcf`: The Probe Card File created by KITT.
- `.gdf`: The Global Data File created by KITT.
- `.psf`: The Parameter Set File created by the Parameter Set Editor (PSE).
- `.klf`: The Keithley Limits File created by the Limits File Editor (LFE).
- `.wpf`: The Wafer Plan File created by the Wafer Plan Editor in the Keithley Test Program Manager (KTPM).
- `.cpf`: The Cassette Plan File created by the Cassette Plan Editor in KTPM.
- `.krf`: The Keithley Recipe File created by the Keithley Recipe Manager (KRM) Tool.
- `.kdf`: The Keithley Data File created by an executed test, to be used by the Keithley Summary Utility (KSU).
- `.uap`: The user access point (UAP) file used by KTPM and the Keithley Test Execution Engine (KTXE).

- `.kpf`: The Keithley Plot File created by KITT, to be used by the Keithley Curve Analysis Tool (KCAT).
- `.c`: The Keithley User Library Tool (KULT) module file. Used and created by KULT.

Wafer description file format

Filename

`wafer_description_file.wdf`

Format

```
#Keithley Wafer Description File
Version,n.n
File,file_path_and_name.wdf
Date,mm/dd/yyyy
Comment,comment string
Project,project_type
DiameterUnits,english_or_metric
Diameter,diameter_measurement
Units,english_or_metric
DieSizeX,x.x
DieSizeY,y.y
Orientation,notch_or_flat,position
WaferOffset,x,y
Axis,axis
Origin,x,y
Target,x.x,y.y
AutoAlignLocation,x.x,y.y
Optimize,optimization_style
RevID,$Revision: n.n $
<EOH>
Pattern,pattern_name
site_or_project_name,x,y
site_or_project_name,x,y
Pattern,pattern_name
site_or_project_name,x,y
site_or_project_name,x,y
<EOSITES>
Site,project_name,probe_description
subsite_name,x.x,y.y
subsite_name,x.x,y.y
<EOSUBSITES>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.wdf` extension) of the Wafer Plan Definition File.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Comment`: May contain any relevant text, up to 256 characters.
- `Project`: Contains the project type (either `Single` or *`Multiple`*).
- `DiameterUnits`: Contains the diameter measurement system (either `English` or `Metric`).
- `Diameter`: Contains the diameter measurement in inches or millimeters, selected from a list of choices as specified in `wdu.ini`.
- `Units`: Contains the die size measurement system (either `English` or `Metric`). This is separate from the diameter units, which means that the wafer diameter can be specified in English units while the rest of the size and position information can be specified in Metric units, and vice-versa.
- `DieSizeX`: Contains the x measurement of the die size in mils or microns. If this is a multiple-project wafer, this field must be 0; site x-y values will be specified in mils or mm instead of die offsets.
- `DieSizeY`: Contains the y measurement of the die size in mils or microns. If this is a multiple-project wafer, this field must be 0; site x-y values will be specified in mils or mm instead of die offsets.
- `Orientation`: Contains the orientation marker (either `Notch` or `Flat`) and the position of the marker (either `Top`, `Bottom`, `Right`, or `Left`).
- `WaferOffset`: Contains two pixels used by the Wafer Description Utility (WDU) only. The wafer offset is for graphics use only; however, it is very important to be able to align the grid on the wafer to match the grid as presented by lithography tools. The maximum X and Y offset values are equal to the die size.
- `Axis`: Contains the proper axis number (`1` to `4`). The different numbers stand for the following orientations:
  - `1` = X, right; Y, up
  - `2` = X, left; Y, up
  - `3` = X, left; Y, down
  - `4` = X, right; Y, down
- `Target`: Contains the target die coordinates for alignment. For multiple projects, this field contains the target offset coordinates for the first probed site.

- `AutoAlignLocation`: Contains the offset coordinates (using the orientation of the `Axis`, above) from the notch or flat, to the alignment die. This is an optional line used to help position the wafer on the prober so the operator must only validate the position and, if necessary, do some fine adjustments. This would be unused for fully automated probers as the alignment information would be in the product files. This value is saved, but not used in the current Keithley Test Execution Engine (KTXE); however, a user access point (UAP) is provided at the point where the alignment of the first wafer should be done.
- `Optimize`: Contains the optimization style as a serpentine pattern number (`1` to `8`). If the wafer is to be probed as listed, `0` should be used in this field.
- `RevID, $Revision: n.n $`: Field for Version Control option, where `n.n` is the revision number for the option.

The header section terminates with the `<EOH>` tag.

Sites

This section contains pattern names with sites. The first line in each pattern defines the pattern name.

- `Pattern`: contains the pattern name. This one-word string must be a valid C-identifier: It must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_).

This line is followed by any number of sites or projects to be associated with the pattern. Each line contains these fields:

- The site name (for single-project wafers) or the project name (for multiple-project wafers) that is to be associated with the pattern.
- The x coordinate.
- The y coordinate.

This section terminates with the `<EOSITES>` tag.

Subsites

This section specifies subsites. Multiple-project wafers can have several such entries, while single-project wafers should have only one. The first line of each entry contains these fields:

- The keyword `Site`.
- The project name. For single-project wafers, this is the string `Single`. For multiple-project wafers, this can be any valid C-identifier as described above, up to 32 characters.
- The project description. For single-project wafers, this should also be the string `"Single"`. For multiple-project wafers, this can be any relevant text.

This line is followed by a list of subsites, one per line, each containing these fields:

- The subsite name. This can be any valid C-style string as described above.

- The x coordinate of the subsite.
- The y coordinate of the subsite.

This section terminates with the <EOSUBSITES> tag.

General Notes

Lines that begin with a # are ignored.

If a file is multiple-project, that means that multiple site types are possible. A site name must be entered for each site to be probed. A list of subsites will be entered for each site type (each site type has a unique site name). Tests will still be bound to subsites. All sites within a probe pattern must contain the same subsites, although they may be in different locations.

There is no graphical representation of multiple-project wafers currently in WDU. The pattern, site, and subsite information is entered directly into the appropriate tables.

Wafer Description Files are generated by WDU, and can be found in the $KI_KTXE_WDF directory.

Test structure file format

Filename

subsite_name.tsf

Format

```
#Keithley Test Structure File
Version,n.n
File,file_path_and_name.tsf
Date,mm/dd/yyyy
Id,id_string
Comment,comment_string
RevID,$Revision: n.n $
<EOH>
devname,STRING,device_name
device_data_name,datatype,value
device_data_name,datatype,value
<EODEV>
devname,STRING,device_name
device_data_name,datatype,value
device_data_name,datatype,value
<EODEV>
<EOTSF>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and file name (with `.tsf` extension) of the Test Structure File. The file name must match a subsite name referenced in the Keithley Test Macro (`.ktm`) and the Wafer Description File (`.wdf`).
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Id`: May contain any relevant text, up to 255 characters.
- `Comment`: May contain any relevant text, up to 255 characters.

The header section terminates with the `<EOH>` tag.

Devices

These sections define the devices of the test structure. Each line contains these fields:

- The data name. This string need not be a valid C-identifier; it may contain nonalphanumeric characters, including spaces. The only constraint is that it may not include commas.
- The data type. All data types defined in `kiox_def.h` are supported. In addition, a special data type, `IDENTIFIER`, may be used to reference a predefined probe card or global identifier.
- The data value. This must be of the type specified in the second field. If it is a `STRING`, the only constraint is that it may not contain commas.

The first line of each section must have `devname` in the first field, `STRING` in the second field, and the device name itself in the third field. This device name must be unique within the test structure (but may be used in other Test Structure Files). Each device section terminates with the `<EODEV>` tag.

General Notes

The Test Structure File terminates with the `<EOTSF>` tag. Lines that begin with a `#` are ignored.

Test Structure File is generated by Test Script Editor (TSE) and can be found in the `$KI_KTXE_TSF` directory.

Keithley test macro (.ktm)

```
/*>> KITT MODULE GENERATION VERSION Vn.n date_and_time */
RevID,$Revision: n.n $ /*n.n is the revision number.*/
/*>> KTM TEST MODULE DESCRIPTION FOR macro_name */
/*
description of the macro
*/
/*>> KTM WAFER & SUBSITE NAME
   wafer_description_file_name
         subsite_name
END KTM WAFER & SUBSITE NAME*/
/*>> KTM PROBE CARD FILE NAME
   probe_card_file__name
   END KTM PROBE CARD FILE NAME*/
/*>> KTM GLOBAL DATA FILE NAME
   global_data_file_path_and_name.gdf
   END KTM GLOBAL DATA FILE NAME*/
/*>> KTM CONSTANT & GENERAL PURPOSE VARIABLES */
/*>> KTM TEST MODULE VARIABLES FOR macro_name */
   datatype variable_name; /* for module_name */
         datatype variable_name; /* for module_name */
   /* Global Pre-Defined Identifiers */
         datatype constant_name = value; /* Constant Declaration */
         datatype constant_name = value; /* Constant Declaration */
   /* Local Pre-Defined Identifiers */
         datatype constant_name = value; /* Constant Declaration */
         datatype constant_name = value; /* Constant Declaration */
/*>> KTM TEST MODULE CONSTANTS SETTINGS
   constant_name, datatype,constant_type,value,
   constant_name, datatype,constant_type,value,
   END CONSTANTS SETTINGS*/

/*>> KTM TEST MODULE PLOT AND LOG ***DO NOT MODIFY***
   variable_name,plot_code,log_code,number,user_code
         variable_name,plot_code,log_code,number,user_code
         END PLOT AND LOG SETTINGS*/
/*>> KTM TEST MODULE BEGIN USRLIB INFORMATION
   user_library_name,user_library_name,
         KTM TEST MODULE END USRLIB INFORMATION*/
/*>> KTM TEST MODULE TEST SEQUENCE FOR macro_name */
   module(param1,param2);
         variable=module(param1,param2);
```

Header

This section contains general information about the file. The entries in this section are marked by the following banners:

- `KITT MODULE GENERATION VERSION`: This banner contains the version number and the date and time the macro was last edited. The date and time take the form `day mon dd tt:tt:tt yyyy` (example: `Mon Jan 01 01:00:00 2000`).
- `KITT TEST MODULE DESCRIPTION`: Contains a description of the macro, which can be any relevant text. The macro name is included in the banner.

- KTM WAFER & SUBSITE NAME: Contains the name of the associated Wafer Description File (without .wdf extension). This may include a path if necessary; if no path is specified, then the file is presumed to be located in the $KI_KTXE_WDF directory.
- KTM PROBE CARD FILENAME: Contains the name of the associated Probe Card File (without .pcf extension). This may include a path if necessary; if no path is specified, then the file is presumed to be located in the $KI_KTXE_PCF directory. This banner and entry are optional, and may be omitted if no .pcf is associated with the macro.
- KTM GLOBAL DATA FILENAME: Contains the name of the associated Global Data File (without .gdf extension). This may include a path if necessary; if no path is specified, then the file is presumed to be located in the $KI_KTXE_GDF directory. This banner and entry are optional, and may be omitted if no .gdf is associated with the macro.

Declarations

This section declares the variables necessary to run the macro, and specifies some important settings. The entries in this section are marked by the following banners:

- KTM CONSTANT & GENERAL PURPOSE VARIABLES: Is a reserved section.
- KTM TEST MODULE VARIABLES: Contains the declarations for the variables necessary to run the macro. The first, untitled segment of this section declares the variables first used in the body of the macro; each declaration is followed by a comment describing which module uses the variable. The second segment, titled Global Pre-Defined Identifiers, declares and initializes the global constants specified under the Pre-Defined Identifiers tab in the Keithley Data Editor. Similarly, the third section, titled Local Pre-Defined Identifiers, declares and initializes the local constants specified under the Pre-Defined Identifiers tab in the Keithley Data Editor. The macro name is included in the banner.
- KTM TEST MODULE CONSTANTS SETTINGS: Contains a list of constants and their properties. Each line contains these fields:
  - The constant name. This string should be a local or global identifier specified under the KTM TEST MODULE VARIABLES banner.
  - The data type. This can be any type specified in kiox_def.h, but it should correspond to the C-type the constant was declared as above.
  - The constant type. This can be either GLOBAL or LOCAL.
  - The value associated with the constant name.
- KTM TEST MODULE PLOT AND LOG: Contains Keithley Interactive Test Tool (KITT) configuration information for each of the test module variables (but not for the constants). Each line in this entry contains these fields:
  - The variable name. This string should be an identifier specified under the KTM TEST MODULE VARIABLES banner.

- The variable plot code. This can be either `PLOT_X` if Keithley Curve Analysis Tool (KCAT) should use the results of this variable as x values, `PLOT_Y` if you want KCAT to use the values of this variable as y values, or `PLOT_OFF` if KCAT should ignore the variable.
- The variable log code. This can be either `LOG_ON` if you want the variable results to be written to a log file, or `LOG_OFF` if the results should not be logged.
- A number.
- The variable user code. This can be either `USER_ON` if you want to see the results of this variable presented in a scrolling window while the test is running, or `USER_OFF` if the results should remain hidden.

Body

This section details the macro routine. Each line is composed of a module call using appropriate parameters. If the module returns a value, then a variable may be used to capture this return value in the form of `variable = module(param1,param2);`. Each line ends with a semicolon.

General Notes

A `.ktx` extension on a Test Macro File means that KITT has tagged the file as containing errors.

Test Macro Files are generated by KITT and can be found in the `$KI_KTXE_KTM` directory.

Probe card file format

Filename

`probecard_file.pcf`

Format

```
#Keithley Probe Card Definition File
Version,n.n
File,file_path_and_name.pcf
Date,mm/dd/yyyy
Id,id_string
Comment,comment_string
RevID,$Revision: n.n $
<EOH>
pin_name,datatype,pin_number
pin_name,datatype,pin_number
<EOPINS>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.pcf` extension) of the Probe Card File.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.

- `Id`: May contain any relevant text, up to 255 characters.
- `Comment`: May contain any relevant text, up to 255 characters.

The header section terminates with the `<EOH>` tag.

Pins

This section maps alphanumeric symbols to actual pin numbers. Each line contains these fields:

- The pin name. This one-word string must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_).
- The data type of the pin. Currently, this is always `INT`, but this field is included here to allow for future support of other data types (example: `Pinlist`).
- The actual pin number.

This section terminates with the `<EOPINS>` tag.

General Notes

Any line that begins with a `#` is ignored.

Probe Card Files are generated by the Keithley Data Editor in KITT and can be found in the `$KI_KTXE_PCF` directory.

Global data file format

Filename

`global_data_file.gdf`

Format

```
#Keithley Global Data Definition File
Version,n.n
File,file_path_and_name.gdf
Date,mm/dd/yyyy
Id,id_string
Comment,comment string
RevID,$Revision: n.n $
<EOH>
data_name,datatype,value
data_name,datatype,value
<EOGDF>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.gdf` extension) of the Global Data File.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Id`: May contain any relevant text.
- `Comment`: May contain any relevant text.

The header section terminates with the `<EOH>` tag.

Data Definitions

This section defines specific variables. Each line contains these fields:

- The data name. This one-word string must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_).
- The data type. All data types defined in `kiox_def.h` are supported.
- The data value. This must be of the type specified in the second field.

This section terminates with the `<EOGDF>` tag.

General Notes

Any line that begins with a `#` is ignored.

Global Data Files are generated by the Keithley Data Editor in KITT and can be found in the `$KI_KTXE_GDF` directory.

Parameter set file format

Filename

`Userlibrary_paramset.psf`

Format

```
#Keithley Parameter Set File
Version,n.n
File,file_path_and_name.psf
Date,mm/dd/yyyy
Id,id_string
Comment,comment string
RevId,$Revision: n.n $
<EOH>
MODULE,module_name
PARAMSET,parameter_set_name
module_paramset_parameter,datatype,value
module_paramset_parameter,datatype,value
<EOPS>
PARAMSET,parameter_set_name
module_paramset_parameter,datatype,value
module_paramset_parameter,datatype,value
<EOPS>
<EOMODULE>
<EOLIB>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.psf` extension) of the Parameter Set File. The filename should have the form *`userlibrary`*`_parmset.psf`, where *`userlibrary`* is the name of the user library the Parameter Set File is associated with.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Id`: May contain any relevant text.
- `Comment`: May contain any relevant text.

The header terminates with the `<EOH>` tag.

Modules

This section defines a module by listing all the parameter sets associated with it. One line is needed before getting into the parameter sets:

- MODULE: Contains the module name. This one-word string must start with a nonnumeric character, and must contain only letters (a through z, A through Z), digits (0 to 9), and the underscore character (_).

This line is followed by any number of Parameter Sets (see information following). This section terminates with the <EOMODULE> tag.

Parameter Sets

These sections define parameter sets. The first line in each section is the PARAMSET line:

- PARAMSET: Contains the name of the parameter set. This one-word string follows the same rules as the module name, above.

This is followed by a list of parameter definitions, one per line. Each line contains these fields:

- The parameter name. This follows the standard naming rules outlined above.
- The data type. All data types defined in kiox_def.h are supported.
- The data value. This must be of the type specified in the second field.

Each Parameter Set terminates with the <EOPS> tag.

General Notes

The Parameter Set File terminates with the <EOLIB> tag. Lines that begin with a # are ignored.

Parameter Set Files are generated by the Parameter Set Editor (PSE) and can be found in the $KI_KTXE_PSF directory.

Parameter limits file format

Filename

`limits_file.klf`

Format

```
#Keithley Parameter Limits File
Version,n.n
File,file_path_and_name.klf
Date,mm/dd/yyyy
Comment,comment_string
RevID,$Revision: n.n $
<EOH>
ID,parameter_id
NAM,parameter_name
UNT,parameter_units
CAT,parameter_category
RPT,y_or_n
CRT,critical_flag
TAR, target_value
AF,abort_destination
AL,abort_action
VAL,val_low, val_high
SPC,spc_low, spc_high
CNT,cnt_low, cnt_high
ENG,eng_low, eng_high
ena,enabled flag
cla,class data
usr1,user data field 1
usr2,user data field 2
usr3,user data field 3
<EOL>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.klf` extension) of the Parameter Limits File.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Id`: May contain any relevant text.
- `Comment:` May contain any relevant text, up to 246 characters. The default comment created by the Limits File Editor (LFE) is `KLF`.

The header section terminates with the `<EOH>` tag.

Limits

These section provides important data about each parameter, including four sets of production limits.

- `ID`: Contains the unique ID of the parameter. This string must be a valid C-identifier. This string may contain a total of 40 characters.
- `NAM`: Contains the name of the parameter. This string need not be a valid C-identifier; it may contain nonalphanumeric characters, including spaces. The only constraint is that it must not include commas. This string may contain a total of 40 characters.
- `UNT`: Contains the units of the parameter. This string need not be a valid C-identifier; it may contain nonalphanumeric characters, including spaces. The only constraint is that it must not include commas. This string may contain a total of 10 characters.
- `CAT`: Contains the category of the parameter. Each category must be a single-word string, but one parameter can belong to several categories, as long as all the categories are listed (comma or whitespace-delimited) on this line, and do not total more than 20 characters.
- `RPT`: Contains either a `Y` for yes or an `N` for no, depending on whether or not the limit is to be included in the lot summary report compiled by the Keithley Summary Utility (KSU).
- `CRT`: Contains `0` (zero) if the parameter is noncritical and any single digit number from `1` to `9` if the parameter is critical; thus, nine different categories of critical parameters can be designated.
- `TAR`: Contains the target value specification for the parameter. Code can use this value to calculate, on a percent under-over basis, the limits for runtime or report comparisons.
- `AF`: Contains the abort action code. If the Keithley Test Execution Engine (KTXE) runs into an abort condition (see `AL`, below), it will go to the next {subsite (`SS`), site (`S`), wafer (`W`), or lot (`L`)} depending upon the code in this field.
- `AL`: Contains one of the limits (`VAL`, `SPC`, `CNT`, `ENG`) to be used for qualifying the parameter value. KTXE will abort if it comes across a data value that fails to satisfy the limit conditions specified in this field.
- `VAL`: Contains the validity limits, typically used to check for faults within the testing system. A low limit and a high limit can be specified.
- `SPC`: Contains the manufacturing standard's limits. A low limit and a high limit can be specified.
- `CTR`: Contains the process control limits. A low limit and a high limit can be specified.
- `ENG`: Contains the engineering limits, typically used by engineers to make sure components meet design standards. A low limit and high limit can be specified.
- `ena`: Contains either a `Y` for yes or an `N` for no, depending on whether or not the parameter is to be included in the adaptive test. Used only with the Adaptive Test option software.
- `cla`: Contains class data. Can be used with the enable flag or as user data.
- `usr1`: Contains user data. Can contain up to 255 characters of information.
- **usr2**: Contains user data. Can contain up to 255 characters of information.

- usr3: Contains user data. Can contain up to 255 characters of information.
- Each parameter's limits section terminates with the <EOL> tag.

General Notes

Lines that begin with a # are ignored.

Parameter Limits Files are generated by the Limits File Editor (LFE) and can be found in the $KI_KTXE_KLF directory.

Wafer test plan file format

Filename

wafer_plan_file.wpf

Format

```
#Keithley Wafer Plan Definition File
Version,n.n
File,file_name.wpf
Date,mm/dd/yyyy
Comment,comment string
Wafer,wafer_file_name.wdf
Limits,limits_file_name.klf
Probe,probe_file_name.pcf
SS_SORT,yes_or_no
RevID,$Revision: n.n $
<EOH>
Siteplan,siteplan_name,description of siteplan
macro_file_name.ktm
macro_file_name.ktm
Siteplan,siteplan_name,description of siteplan
macro_file_name.ktm
macro_file_name.ktm
<EOSP>
siteplan_name,probe_pattern
siteplan_name,probe_pattern
macro_file_name.ktm,probe_pattern
macro_file_name.ktm,probe_pattern
<EOW>
```

Header

This section contains general information about the file.

- Version: Contains the current version number.
- File: Contains the filename (with .wpf extension) of the Wafer Plan Definition File. This may be preceded by the path to the file, but if no path is specified the file is assumed to reside in the $KI_KTXE_WPF directory.
- Date: Contains the date the file was last edited, in Y2K-compliant form.

- `Comment`: May contain any relevant text, up to 255 characters.
- `Wafer`: Contains the name (with `.wdf` extension) of the appropriate Wafer Description File. This may be preceded by the path to the file, but if no path is specified the file is assumed to reside in the `$KI_KTXE_WDF` directory. This line allows pin mappings to be set for each wafer. This line is ignored if the Cassette Plan File specifies a Wafer Description File.
- `Limits`: Contains the name of the appropriate Parameter Limits File (with `.klf` extension). This may be preceded by the path to the file, but if the path is not specified the file is assumed to reside in the `$KI_KTXE_KLF` directory. This line is optional.
- `Probe`: Contains the name (with `.pcf` extension) of the appropriate Probe Card File. This may be preceded by the path to the file, but if no path is specified the file is assumed to reside in the `$KI_KTXE_PCF` directory. This line allows the probed locations to be set for each wafer. This line is ignored if the Cassette Plan File specifies a Probe Card File. This line is optional.
- `SS_SORT`: Contains either `YES` or `NO`, depending on whether the subsites should be sorted.

The header section terminates with the `<EOH>` tag.

Site plans

This section defines site plans. Each site plan entry contains these fields:

1. The keyword `Siteplan`.
2. The site plan name. This one-word string must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_).
3. A description of the site plan.

This line is followed by a list of Test Macro File names (with `.ktm` extension) to be included in the site plan, one per a line. Each of these may be preceded by the path to the file, but if no path is specified the file is assumed to reside in the `$KI_KTXE_KTM` directory. This section terminates with the `<EOSP>` tag.

Probe patterns

This section associates probe patterns with site plans. Each line has these fields:

1. The name of the object to associate with the probe pattern. This can be either the name of a site plan (as defined above) or a name and optional path of a Test Macro File (with `.ktm` extension) not already associated with a site plan.
2. The name of a probe pattern.

This section terminates with the `<EOW>` tag.

General notes

Lines that begin with a # are ignored.

The Execution Engine reads the wafer plans specified in the Cassette Plan File. The Wafer Plan File contains abort condition checks from the .klf, pin mapping from the .pcf, and the macro-to-probe-pattern mapping information. The probe pattern and subsite position information for this Wafer Plan File is contained in the .wdf.

Wafer Plan Files are generated by the Wafer Plan Editor in Keithley Test Program Manager (KTPM), and can be found in the $KI_KTXE_WPF directory.

Cassette test plan file format

Filename

cassette_plan_file.cpf

Format

```
#Keithley Cassette Plan Definition File
Version,n.n
File,file_name.cpf
Date,mm/dd/yyyy
Comment,comment string
Data,lot_id
Engine,execution_engine
Probe,probe_card_file_name.pcf
Wafer,wafer_card_file_name.wdf
Global,global_data_file_name.gdf
UAPdefaults,uap_defaults_file_name.uap
RevID,$Revision: n.n $
<EOH>
slot_id,wafer_id,wafer_plan_name.wpf
slot_id,wafer_id,wafer_plan_name.wpf
<EOS>
uap_name,library_name,uam
uap_name,library_name,uam
<EOUAP>
```

Header

This section contains general information about the file.

- Version: Contains the current version number.
- File: Contains the filename (with .cpf extension) of the Cassette Plan File. This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the $KI_KTXE_CPF directory.
- Date: Contains the date the file was last edited, in Y2K-compliant form.
- Comment: May contain any relevant text, up to 256 characters.

- `Data`: Contains the lot ID. If no output filename is specified at runtime, then the program will create a file named *`lot_id`*`.kdf` (where *`lot_id`* is specified here) in the `$KI_KTXE_KDF` directory and use that as the output file.
- `Engine`: Contains the name of the execution engine to be used. There can be more than one test execution engine; however, most customer sites should require only one. For example, there could be a production engine and a development engine; however, user access modules (UAMs) could be used to distinguish these different modes of operation...switches are available to change what a single Keithley Test Execution Engine (KTXE) will do for a specific run.
- `Probe`: Contains the name of the appropriate Probe Card File (with `.pcf` extension). This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the `$KI_KTXE_PCF` directory. This line is optional; if the `.pcf` is specified here it will override any `.pcf` files specified in any wafer plan files.
- `Wafer`: Contains the name of the appropriate Wafer Description File (with `.wdf` extension). This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the `$KI_KTXE_WDF` directory. This line is optional; if the `.wdf` is specified here it will override any `.wdf` filename specified in the `.wpf`.
- `Global`: Contains the name of the appropriate Global Data File (with `.gdf` extension). This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the `$KI_KTXE_GDF` directory. This line is optional. Global data can be specified in a `.gdf` file. This information will be created with the Keithley Data Editor (KDE) and read in by the KTXE. The data precedence is set to:
  - PDI - local scope
  - PCF data
  - GDF data
  - PDI - global scope

  Multiple `.gdf` files can be selected for a single `.cpf`.
- `UAPdefaults`: Contains the name of the appropriate User Access Points Defaults File (with `.uap` extension). This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the `$KI_KTXE_UAP` directory. This line is optional. This mechanism will allow multiple UAMs to be grouped into a single file of type `.uap`. This file can be created with any text editor. Each line in the file must conform to the syntax as specified in the user access point (UAP) section of the `.cpf` file.

The header section terminates with the `<EOH>` tag.

Slots

This section associates slots with Wafer Plan Files. Each line contains these fields:

- The slot ID. There are several options here. If the ID is absolute, then this field should only contain the slot number. If the ID is relative, than this field should contain the character `R` immediately followed by the slot number. All the slots can be referenced at one time here by using the keyword `ALL`. Finally, if the keyword `OPR` is used here, then the operator will be prompted to specify the slot during the test execution.
- The wafer ID. This one-word string may contain up to 32 characters, must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_). If the first field reads `ALL` or `OPR`, this field should be left blank.
- The filename of the appropriate Wafer Plan File (with `.wpf` extension). This may be preceded by the path to the file, but if no path is specified then the file is assumed to reside in the `$KI_KTXE_WPF` directory.

It must be noted that having multiple wafer plans will have an impact on the Keithley Summary Utility (KSU). Summaries could be per wafer plan (for example, there could be different parameters or different limits). If the same wafer plan will be used for all the wafers in a cassette then only one line is needed. This line must have the keyword `All` in the first field, a blank second field, and the path and filename of the designated Wafer Plan File. Absolute and relative slot references cannot be used in the same file (the prober must be able to load the wafers in order). This section terminates with the `<EOS>` tag.

User access points

This section modifies the Execution Engine at designated user access points (UAPs). Each line contains these fields:

1. The UAP name. This is the string that identifies the UAP. The `ktpm.ini` file contains lists of the UAPs that are included in each of the available test execution engines.
2. The library name.
3. The user access module (UAM). This can be either a module in the user library specified in the second field, an expression of the form *`return_value=module_name`*`(`*`param1`*`,`*`param2`*`)` using said module, or the name of a Test Macro File (with `.ktm` extension). If it is the latter, the second field should be left empty.

This section allows you to extend (customize) KTXE by creating code to be inserted at designated spots (UAPs) by KTXE. You can specify the modules (UAMs) to be included at any UAPs by modifying this section. This section terminates with the `<EOUAP>` tag.

General Notes

Lines that begin with a `#` are ignored.

Cassette Plan Files are created by the Cassette Plan Editor in Keithley Test Plan Manager (KTPM) and can be found in the `$KI_KTXE_CPF` directory.

The Keithley Data File (KDF) library is being modified to support user-defined data. This information is contained within the KDF file with the following format:

```
<TAG>"tagName",tag value string
```

- The `<TAG>` field is required.
- The `"tagName"` field can contain any characters except the double-quote character. The maximum length of the `tagName` is `PARAM_ID_LENGTH` (128) characters.
- The comma is required and separates the `tagName` field from the tag value string field.
- The `tag value string` has a maximum length of 512 characters and cannot contain the new-line
  character.

Keithley data file format

Filename

`data_file.kdf`

Format

```
TYP,file_typ
LOT,lot_name
PRC,process_name
DEV,device_name
TST,test_name
SYS,system_name
TSN,test_station_id_string
OPR,operator_name_string
STT,dd,mmm,yyyy, tt:tt
SK1,usr_data_1
SK2,usr_data_2
SK3,usr_data_3
LMT,limit_file_name
WDF,wafer_description_file_name
COM,comment_string
<EOH>
wafer_id,wafer_split,wafer_boat,wafer_slot
site_id,row,column
param_id,value
param_id,value
<EOS>
site_id,row,column
param_id,value
param_id,value
<EOS>
param_id,value
param_id,value
<EOS>
```

```
<EOW>
wafer_id,wafer_split,wafer_boat,wafer_slot
site_id,row,column
param_id,value
param_id,value
<EOS>
site_id,row,column
param_id,value
param_id,value
<EOS>
<EOW>
```

Header

This section contains general information about the file.

- `TYP`: Contains the type of the file; typically reads KDF `Vn.n`, where `n.n` is the version number.
- `LOT`: Contains the name of the lot. This string may contain up to 50 characters.
- `PRC`: Contains the process name. This string may contain up to 50 characters.
- `DEV`: Contains the device name. This string may contain up to 50 characters.
- `TST`: Contains the test name. This string may contain up to 255 characters.
- `SYS`: Contains the system name. This string may contain up to 20 characters.
- `TSN`: Contains the test station ID integer (1 to 4).
- `OPR`: Contains the operator name. This string may contain up to 30 characters.
- `STT`: Contains the date and time the file was created, in Y2K-compliant form.
- `SK1`: Contains a user search key. This string may contain up to 30 characters.
- `SK2`: Contains a user search key. This string may contain up to 20 characters.
- `SK3`: Contains a user search key. This string may contain up to 10 characters.
- `LMT`: Contains the Parameter Limits File name. This string may contain up to 80 characters.
- `WDF`: Contains the Wafer Description File name. This string may contain up to 80 characters.
- `COM`: May contain any relevant text, up to 256 characters.

The header terminates with the `<EOH>` tag.

Wafers

These sections list all the sites within a given wafer. One line is needed before getting into the site sections, and that line contains these fields:

- The wafer ID. This one-word string must be a valid C-style identifier; it must start with a nonnumeric character, and must contain only letters (`a` through `z`, `A` through `Z`), digits (`0` to `9`), and the underscore character (_).

- The wafer split. This optional one-word string must be a valid C-style identifier as described above.
- The wafer boat. This optional one-word string must be a valid C-style identifier as described above.
- The wafer slot. This one-word string must be a valid C-style identifier as described above.

This line is followed by any number of sites (see below). Each wafer section terminates with the `<EOW>` tag.

Sites

These sections report the data for each site. The first line in each section contains these entries:

- The site ID. This one-word string must be a valid C-style identifier as described above.
- The site row number.
- The site column number.

This line is followed by a list of all the measurements that were taken for that site, one per line. Each line contains these fields:

- The parameter ID. This is the C-style identifier string that matches a parameter ID in the Parameter Limits File.
- The numeric value of the measurement.

Each site section terminates with the `<EOS>` tag.

General notes

- Lines that begin with a `#` are ignored.
- Data files are generated by test executions. These files are in the `$KI_KTXE_KDF` directory.

User access point file format

Filename

`user_access_point_file.uap`

Format

```
#Keithley UAP File
Version,n.n
File,file_path_and_name.uap
Date,mm/dd/yyyy
Id,id_string
Comment,comment_string
RevID,$Revision: n.n $
<EOH>
uap_name,user_library_name,routine_name_or_expression
uap_name,user_library_name,routine_name_or_expression
<EOUAP>
```

Header

This section contains general information about the file.

- `Version`: Contains the current version number.
- `File`: Contains the path and filename (with `.uap` extension) of the UAP File.
- `Date`: Contains the date the file was last edited, in Y2K-compliant form.
- `Id`: May contain any relevant text.
- `Comment`: May contain any relevant text.

The header section terminates with the `<EOH>` tag.

User access points

This section defines the user access points (UAPs). Each line contains these fields:

- The UAP name.
- The user library name of the module you want to use at this UAP.
- A routine name or an expression using the routine name. If an expression is used, it must be of the form `return_value = routine_name(param1,param2)`.

This section terminates with the `<EOUAP>` tag.

General Notes

Any line that begins with a `#` is ignored.

Keithley plot file format

Filename

`plot_file.kpf`

Format

```
#<KTE>Keithley Results File
#<VERSION>n.n
#<DELIMITER>,
parameter1_id,parameter2_id,parameter3_id,
site1_parameter1_data,site1_parameter2_data,site1_parameter3_data,
site2_parameter1_data,site2_parameter2_data,site2_parameter2_data,
```

Header

This section contains general information about the file.

- `VERSION`: Contains the current version number.
- `DELIMITER`: Contains a single character (usually `,`) to serve as a delimiter while listing the data points.

Every line in this section begins with a `#`.

Parameters

This section is just one line long. It lists the variables specified in in the Keithley Interactive Test Tool (KITT), separated by the delimiter specified in the header. The delimiter also ends the line.

Data

This section lists the actual data values for each parameter above. Each line represents one site worth of data values. The values are simply written out in the same order as the parameter section, separated by the delimiter specified in the header. The delimiter also ends the line.

General Notes

Plot files are generated by the Results Window in KITT, and can be found in the `$KI_KTXE_KDF` directory.

KULT module file format

Filename

`kult_module_file_name.c`

Format

```
/* USRLIB MODULE INFORMATION
   MODULE NAME: kult_module_name
   MODULE RETURN TYPE: datatype
   NUMBER OF PARAMS: n
   ARGUMENTS:
   param, datatype, input_or_output, default, min , max
   param, datatype, input_or_output, default, min, max
#include <header_file_name.h>
#include <header_file_name.h>
END USRLIB MODULE INFORMATION
*/
   /* USRLIB MODULE HELP DESCRIPTION
   END USRLIB MODULE HELP DESCRIPTION */
/* USRLIB MODULE VERSION CONTROL */
static char const vcid[] ="$Id: kult_module_name.c,v 1.2 2000/09/27 14:09:43 user Exp
   $";
/* USRLIB MODULE PARAMETER LIST */
   #include <header_file_name.h>
   #include <header_file_name.h>
   datatype kult_module_name( datatype param, datatype param )
{
          /* USRLIB MODULE CODE */
   code_body
   /* USRLIB MODULE END */
}
/*End kult_module_name.c */
```

INCLUDES:

Header

This section contains general information about the file. There are two banners in this header. The first banner reads `USRLIB MODULE INFORMATION`, and contains the following entries:

- `MODULE NAME`: Contains the name of the module (the Keithley User Library Tool (KULT) module file name itself should be the name of the module with `.c` extension.)
- `MODULE RETURN TYPE`: Contains the data type of the module's return value. This is restricted to the following types: `char`, `float`, `double`, `int`, `long`, and `void`.
- `NUMBER OF PARAMS`: Contains the number of parameters the module takes.

- ARGUMENTS: Details information about each of the parameters, one per line. Each line contains these fields:
  - The name of the parameter. This one-word string should be a valid C-style identifier; it must start with a nonnumeric character, and must contain only letters (a through z, A through Z), digits (0 to 9), and the underscore character (_).
  - The data type of the parameter. This can be any of (or a pointer to any of) the following types: char, double, float, int, long. Three array types are also available: F_ARRAY_T (for floats), D_ARRAY_T (for doubles), and I_ARRAY_T (for ints). If an array type is used, then the next line must be the entry for the number of elements in this array, and should have the form ArrSizeForarray_param, int, Input, default, min, max, where *array_param* is the name of the array.
  - Either Input or Output, depending on the purpose of the parameter.
  - The default value of the parameter. This field is optional, but if it is used then the type should match the data type in the second field. If the type in the second field is an array, then this field should be left empty.
  - The minimum value of the parameter. This field is optional, but if it is used then the type should match the data type in the second field. If the type in the second field is an array, then this field should be left empty.
  - The maximum value of the parameter. This field is optional, but if it is used then the type should match the data type in the second field. If the type in the second field is an array, then this field should be left empty.
- INCLUDES: Lists all of the header files that are to be included in the module, in the standard C compiler-directive format.

The next banner reads USRLIB MODULE HELP DESCRIPTION, which may be followed by any number of lines of text commentary about the module.

Include directives

This section actually instructs the compiler to include the header files listed in the header section. After the banner USRLIB MODULE PARAMETER LIST, the include files should be listed exactly the way they appear in the header section.

Code body

This section contains the actual code used by the compiler. The function header (not to be confused with the header section as detailed above) should include all of the parameters listed in the header section. If an array is declared in the header section, then the parameter should be listed in the function header as a pointer to the type of array declared (double, float, or int). The function header is followed by a pair of curly brackets; between them is all of the code necessary to execute the module.

General Notes

Every KULT Module File is a C-language document, but not every C-language document is a KULT Module File. To be a KULT Module File, the C-language document must conform to the format explained above.

Test Macro Files are generated by KULT and can be found in the `$KI_KULT_PATH/usrlib` directory, where `usrlib` is the name of the module's user library.

Data pool

The data pool is used to hold global data while the Keithley Test Execution Engine (KTXE) is running. When KTXE is started, the variables declared in the global data files and probe card file are copied into the data pool. The data pool is accessible at any point during test execution. This allows you to access global variables in test macros generated in the Keithley Interactive Test Tool (KITT) and also pass data pool items as parameters to modules generated from the Keithley User Library Tool (KULT) that are run at user access points (UAPs).

For example, you could be using global data in your Keithley Test Module (KTM). When KTXE executes a test macro and encounters a variable, it checks for the variable in the following order:

1. KTXE checks the variable against the list of Pre-Defined Identifiers (PDIs) of local scope. If the variable is found, it is passed on to the test macro.
2. If the variable is not found in the local PDIs, the data pool is then searched and the variable is passed on to the test macro.

The data pool is also used to look up variables for parameters that are passed to KULT-generated modules when they are run at the UAPs. When you select UAPs in Keithley Test Program Manager (KTPM) and start the user access module selector to choose the KULT module, you will have to enter the values for each of the parameters of the module. For the values of the parameters, you can type in absolute values or specify a variable that is in the data pool. When KTXE processes the UAPs, it checks to see if the parameter being passed to the module is an absolute or a variable value. If a variable is being passed, the value of the variable is looked up in the data pool.

Since the data pool holds global data, the variables in the data pool are updated as they are changed. For example, if KTXE executes a test macro that contains instructions that modify the value of a variable in the data pool (for example, a data pool variable is used as a return value variable), the data pool is updated as soon as the value of the variable is changed. This allows the results generated from the execution of one macro to be used in another macro or at a UAP.

KTXE puts the items listed in the following tables into the data pool.

KI_ktxe_redo_macro

Name	Type	Initial value	First available	Last usable
KI_ktxe_redo_macro	**INT**	**0**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
Flag that allows a macro to be re-executed. Set this at or before UAP_TEST_END or UAP_TEST_DATA_LOG to retest. Any non-zero value means retest. This flag is reset automatically to zero once it is set. Multiple retests will require the flag to be set each time.				

KI_ktxe_retest_wafer

Name	Type	Initial value	First available	Last usable
KI_ktxe_retest_wafer	**INT**	**0**	**UAP_WAFER_BEGIN**	**UAP_WAFER_END**
Flag that allows a wafer to be retested. Set this at or before UAP_WAFER_END to retest. Any non-zero value means retest. This flag is reset automatically to zero once it is set. Multiple retests will require the flag to be set each time.				

KI_ktxe_skip_limits_check

Name	Type	Initial value	First available	Last usable
KI_ktxe_skip_limits_check	**INT**	**0**	**UAP_TEST_DATA_LOG**	**UAP_TEST_DATA_LOG**
If set to 1, causes KTXE to skip the default limit/results checking for abort conditions. The user can use their own custom checking routine at UAP_TEST_END, or UAP_TEST_DATA_LOG. If custom routines are used, this flag MUST BE SET at or before UAP_TEST_DATA_LOG, else the default limit check routine could/would reset the abort flag(s).				

KUI_User

Name	Type	Initial value	First available	Last usable
KUI_User	**LONG_P**	**&KUI_User**	**UAP_PROG_ARGS**	**UAP_LOT_INFO**

Pointer to the KUI_User structure. This allows access to two user-defined fields on the KTXE Status Dialog window. The KUI_User structure is initialized to NULL by default, causing the user fields to be invisible. Initializing the "label_1" or "label_2" fields at or before UAP_LOT_INFO, will enable the user fields and make them visible on the Status Window.

Example:

```
#include "kui_proto.h"
#include "COM_usrlib.h"

kui_user_t *user ;
long *tmp ;

tmp = ( long *)dpGetPointer( "KUI_User", LONG_P ) ;
if ( tmp == NULL )
  return ;

user = ( kui_user_t *)*tmp ; /* <---- VERY IMPORTANT to de-reference tmp */

/* Initialize field 1 */
strcpy( user->label_1, "First Label " ) ;

      strcpy( user->data_1, "Data for first field" ) ;


/* Initialize field 2 */
strcpy( user->label_2, "Second Label " ) ;
strcpy( user->data_2, "Data for second field" ) ;

KTXEUpdateStatusAbort( "Forcing update of display..." ) ;
```

ManualProberType

Name	Type	Initial value	First available	Last usable
ManualProberType	**INT**	**0**	**UAP_PROG_ARGS**	**UAP_PROBER_INIT**

Flag to tell KTXE that we have a CM61-type prober that will allow a PrLoad command. You must set this value via a Global Data file or at UAP_PROG_ARGS in order for this to take effect.

ManualWaferLoad

Name	Type	Initial value	First available	Last usable
ManualWaferLoad	**INT**	**0**	**UAP_PROG_ARGS**	**UAP_PROBER_INIT**

Flag to tell KTXE that we have a prober that allows front-access loading of single wafers. You must set this value via a Global Data file or at UAP_PROG_ARGS in order for this to take effect.
Example: You have an EG4080 prober and want to load a single wafer via the front access tray. Set the ManualWaferLoad flag to 1 in a global data file referenced by your cassette plan and run KTXE with said cassette plan. You will be prompted to manually load/profile/align the wafer and press OK. The wafer will be tested normally. You will then be prompted to manually unload the wafer.

UAP_abort_level

Name	Type	Initial value	First available	Last usable
UAP_abort_level	**INT**	**uap_abort_level**	**after a UAP is executed**	**after a UAP is executed**
Return value of a UAP routine if assigned in KTPM. Can be used to cause KTXE to abort after a UAP is executed. If the UAP returns the value KI_ABORT, KTXE will exit. **Example** A UAP description within a cassette plan file as: UAP_TEST_END,myLib,UAP_abort_level=MyUAProutine() will cause KTXE to exit if MyUAProutine returns KI_ABORT.				

abort_code

Name	Type	Initial value	First available	Last usable
abort_code	**INT**	**-1**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
abort_code_description	**CHAR_P**	**&abort_code_description**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
These two values are used by the UAP_ABORT_EXIT_HDLR routines. They can be set to user-defined values for abort handing.				

abort_level

Name	Type	Initial value	First available	Last usable
abort_level	**INT_P**	**&abort_level;**	**UAP_SITE_CHANGE**	**UAP_ENGINE_EXIT**

Value that determines the abort level of a limit.

Example:

```
int *abort_level ;
enum abort_level_t abort_code ;
/* something happens to require the abort_code to be set to one
 * of the following:
 */
abort_code = NOABORT ;
abort_code = SUBSITEABORT ;
abort_code = SITEABORT ;
abort_code = WAFERABORT ;
abort_code = LOTABORT ;
/* Check abort code and react accordingly
* set the abort level flag for the proper skip action
* (the execution engine will handle the abort just as if a limit/result
* pair had caused an abort...)
*/
abort_level = ( int *)dpGetPointer( "abort_level", INT_P ) ;
*abort_level = abort_code ;
if ( NOABORT == abort_code )
 return ;
/* we are here so this means we need to skip something...
 * Skip the "normal" limit/result list abort checking so we don't
 * reset the abort code that we just set up!!
 */
dpAddData( "KI_ktxe_skip_limits_check", INT, 1 ) ;
```

cl_kwf_fname

Name	Type	Initial value	First available	Last usable
cl_kwf_fname	**CHAR_P**	**cl_kwf_fname**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
Wafer description filename entered from "-w" command-line switch.				

confirm_oper_wafers

Name	Type	Initial value	First available	Last usable
confirm_oper_wafers	**INT**	**0**		
If this value is set to 1 and the operator selects an empty slot for OPERATOR MODE, an error message will be displayed. This value is set to off by default.				

cpf_info

Name	Type	Initial value	First available	Last usable
cpf_info	**LONG_P**	**cpf_info**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**

Pointer to the cassette plan info structure.

```
typedef struct
{
        char version[MAXVERSIZE];
        char cpfname[MAXFILENAMESIZE];
        char ascdatetime[MAXDATESTRSIZE];
        char comment[MAXCOMMENTSTRSIZE];
        char uapdefaults[MAXENGNAMESTRSIZE];
        char engine[MAXENGNAMESTRSIZE];
        char pcffname[MAXFILENAMESIZE];
        gdffname_list_t *gdffname_list;
        char kdffname[MAXFILENAMESIZE];
        char wdffname[MAXFILENAMESIZE];
        slot_list_t *slot_list;
        uap_list_t *uap_list;
}
cpf_info_t;
```

current_slot_list

Name	Type	Initial value	First available	Last usable
current_slot_list	**LONG_P**	**& current_slot_list**	**UAP_LOT_INFO**	**UAP_ENGINE_EXIT**

Pointer to the current_slot_list element. Updated each wafer.
Example:

```
long *tmp ;
slot_list_t *slot ;
/* Get the current slot list pointer..
 */
tmp = ( long *)dpGetPointer( "current_slot_list", LONG_P )  ;
slot = ( slot_list_t *) *tmp ;
```

Structure definition:

```
typedef struct _slot_list
{
        char        slot[SLOTNAMESIZE];
        char        wafer_id[MAXWAFERID];
        char        split_id[MAXWAFERID];
        char        wpfname[MAXFILENAMESIZE];
        char        plan_wpfname[MAXFILENAMESIZE];
        struct _slot_list *next;
}
slot_list_t ;
```

current_wwp_list

Name	Type	Initial value	First available	Last usable
current_wwp_list	**LONG_P**	**current_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**

Pointer to the current node of the working wafer plan (WWP) list. Updated each Macro.
Structure definition: See type definition for wwp_list_t in $KIHOME/include/ktxe_types.h.

cur_wwp_list_ptr

Name	Type	Initial value	First available	Last usable
cur_wwp_list_ptr	**LONG_P**	**&current_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**

This is a pointer to the `current_wwp_list` pointer value. This allows execution flow adjustment by manipulation of the `current_wwp_list` pointer value.
Example:

```
wwp_list_t *wwp ;
long *foo ;
foo = ( long *)dpGetPointer( "cur_wwp_list_ptr", LONG_P ) ;
wwp = ( wwp_list_t *) *foo  ;
/* Skip until Site_3 node
*/
if( wwp->next != NULL )
{
        while ( wwp->next != NULL )
        {
                if ( strcmp( wwp->next->siteid, "Site_3" ) == 0 )
                        break ;
                else
                        wwp = wwp->next  ;
        }
}
/* adjust the current_wwp_list value
 */
*foo = (long) wwp ;
```

display_lotdlg

Name	Type	Initial value	First available	Last usable
display_lotdlg	**INT_P**	**&display_lotdlg**	**UAP_PROG_ARGS**	**UAP_LOT_INFO**

Flag to enable/disable Lot Dialog display. Defaults to 1, or enabled. MUST be updated at or before UAP_LOT_INFO.
Example:

```
int *logDlg ;
/* disable the display of the Lot Dialog screen
 */
lotDlg = ( int *)dpGetPointer( "display_lotdlg", INT_P ) ;
*lotDlg = 0 ;
```

failed_result_list

Name	Type	Initial value	First available	Last usable
failed_result_list	**LONG_P**	**failed_result_list**	**UAP_HANDLE_ABORT**	**UAP_HANDLE_ABORT**

Pointer to a results list containing results that failed the limits checking. Use this for any custom abort handling requirements.

ktm_list

Name	Type	Initial value	First available	Last usable
ktm_list	**LONG_P**	**ktm_list_head**	**UAP_VALIDATE_OCR**	**UAP_WAFER_END**

Pointer to the list of all KTMs used in a wafer plan. This is updated each wafer before UAP_WAFER_BEGIN.
Structure definition:

```
typedef struct _ktm_list
{
        char ktmfname[MAXFILENAMESIZE];
        char wafpatname[MAXWAFPATNAMESIZE];
        struct _ktm_list *next;
}
ktm_list_t;
```

ktxe_abort_logging

Name	Type	Initial value	First available	Last usable
ktxe_abort_logging	**INT_P**	**&ktxe_abort_logging**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**

Flag to enable logging of the limits abort.

ktxe_cpf_mode

Name	Type	Initial value	First available	Last usable
ktxe_cpf_mode	**INT**	**cassette plan probe mode**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**

Flag that indicates the probe mode of the cassette plan. This value is set when the Cassette plan is loaded. This flag can be useful for Prober error recovery sequencing.
ALL_MODE = 1 OPR_MODE = 2 ABS_MODE = 3 REL_MODE = 4

ktxe_debug

Name	Type	Initial value	First available	Last usable
	INT_P	**&ktxe_debug**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**

Debug flags from "-x" command-line switch.

ktxe_disable_exec_log

Name	Type	Initial value	First available	Last usable
ktxe_disable_exec_log	**INT**	**undefined**		

Flag that prevents the execution log from being placed into KDF file. Define flag (set to 1) at or before UAP_WRITE_LOT_INFO. This data pool switch is only effective while KTXE is executing a recipe.

ktxe_disable_kdf

Name	Type	Initial value	First available	Last usable
ktxe_disable_kdf	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_WAFER_END**
Flag for disabling standard kdf data logging.				

ktxe_disable_ktm

Name	Type	Initial value	First available	Last usable
ktxe_disable_ktm	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_LOT_END**
Flag for disabling macro test execution.				

ktxe_disable_kui

Name	Type	Initial value	First available	Last usable
ktxe_disable_kui	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_LOT_END**
Flag for disabling user interface for KTXE.				

ktxe_disable_load_cassette_msg

Name	Type	Initial value	First available	Last usable
ktxe_disable_load_cassette_msg	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_PROBER_INIT**
Flag for disabling "Load Cassette" message dialog.				

ktxe_disable_plan_complete_msg

Name	Type	Initial value	First available	Last usable
ktxe_disable_plan_complete_msg	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
Flag for disabling the "KTXE Plan Complete" message.				

ktxe_disable_prober

Name	Type	Initial value	First available	Last usable
ktxe_disable_prober	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_WAFER_END**
Flag for disabling default KTXE prober routines.				

ktxe_disable_prober

Name	Type	Initial value	First available	Last usable
ktxe_disable_prober	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_WAFER_END**
Flag for disabling default KTXE prober routines.				

ktxe_disable_statdlg

Name	Type	Initial value	First available	Last usable
ktxe_disable_statdlg	**INT**	**0**	**UAP_PROG_ARGS**	**UAP_LOT_INFO**
Flag for disabling KTXE status dialog window. Set this flag to 1 to disable the KTXE status dialog window or add to a global data file used by the recipe or cassette plan.				

ktxe_disable_uap

Name	Type	Initial value	First available	Last usable
ktxe_disable_uap	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
Flag for disabling execution of UAP routines.				

ktxe_enable_cl_log

Name	Type	Initial value	First available	Last usable
ktxe_enable_cl_log	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_POST_LOT_INFO**
Flag for enabling/disabling logging of command line arguments. Set this flag to 1 to log KTXE command-line arguments to the KDF file. Set this flag to 0 to prevent logging. Defaults: Cassette plan mode: 0 (prevents logging); Recipe mode: 1 (allows logging). A global data file or a UAP routine can be used to alter the state of this variable.				

ktxe_disable_exec_log

Name	Type	Initial value	First available	Last usable
ktxe_disable_exec_log	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_POST_LOT_INFO**
Flag for disabling logging of list of files. Set this flag to 1 to prevent list of files used by a recipe from being logged into the KDF file.				

ktxe_enable_doc

Name	Type	Initial value	First available	Last usable
ktxe_enable_doc	**INT**	**FALSE**	**UAP_PROG_ARGS**	**UAP_PROG_ARGS**
Flag for enabling test execution documentation data collection.				

ktxe_error_gui

Name	Type	Initial value	First available	Last usable
ktxe_error_gui	**INT**	**0**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
Flag for defining the behavior of KTXE when an error occurs. Default, (ktxe_error_gui undefined or set to other than 1 or 2), errors are written to the error log. If ktxe_error_gui == 1; error msg will be displayed and the user has the option to continue or quit. If ktxe_error_gui == 2; error msg will be displayed and the user can only quit.				

ktxe_fill_opr_dlg

Name	Type	Initial value	First available	Last usable
ktxe_fill_opr_dlg	**INT**	**0**	**UAP_PROG_ARGS**	**UAP_PROBER_INIT**
Flag for enabling/disabling the initialization of the Wafer ID dialog window. If this flag is set to 1, the Wafer ID dialog window will be filled with the wafers available from the PrCassetteMap call. If this flag is set to 0, the Wafer ID dialog window will be empty. Default state is 0.				

ktxe_min_SS_touch

Name	Type	Initial value	First available	Last usable
ktxe_min_SS_touch	**INT**	**ktxe_min_SS_touch**	**UAP_PROG_ARGS**	**UAP_WAFER_PREPARE**
Flag that enables/disables subsite touchdown minimalization to occur. Default state is 1, enabled. This flag is ONLY effective when the ktxe_sort_subsite_ktms flag is disabled. If the wafer plan file specified multiple macros at a subsite, and the macros are not listed together, this flag will determine if the probes will touch-down multiple times at the same subsite, or only once. Must be updated at or before UAP_WAFER_PREPARE.				

ktxe_missingSS_ok

Name	Type	Initial value	First available	Last usable
ktxe_missingSS_ok	**INT**	**0**		
Set to non-zero to suppress the "missing Subsite error" messages. This will allow the KTDT Validation step to pass without error.				

ktxe_reload_wafer_plan

Name	Type	Initial value	First available	Last usable
ktxe_reload_wafer_plan	**INT**			**UAP_WAFER_END**
Wafer plan reloading may be forced by defining this data pool item. If a working wafer plan (WWP) list has been modified, it will be necessary to define this item to force the next WWP list to be reset to the original. This item must be added at or before each UAP_WAFER_END. This item is removed at the end of each wafer loop.				

ktxe_report_no_klf

Name	Type	Initial value	First available	Last usable
ktxe_report_no_klf	**INT**	**1**	**UAP_PROG_ARGS**	**UAP_WAFER_PREPARE**
This flag defaults to value 1 and if set to 0 will cause KTXE to suppress the generation of a "Missing Limits File" error message.				

ktxe_report_no_pcf

Name	Type	Initial value	First available	Last usable
ktxe_report_no_pcf	**INT**	**1**	**UAP_PROG_ARGS**	**UAP_WAFER_PREPARE**
This flag defaults to value 1 and if set to 0 will cause KTXE to suppress the generation of a "Missing Probe Card File" error message.				

ktxe_sort_subsite_ktms

Name	Type	Initial value	First available	Last usable
ktxe_sort_subsite_ktms	**INT**	**ktxe_sort_subsite_ktms**	**UAP_PROG_ARGS**	**UAP_WAFER_PREPARE**
Flag that enables/disable subsite sorting to occur. Default state is 1, enabled. Disabling this flag will cause macros to be executed in the order specified by the wafer plan file. Must be updated at or before UAP_WAFER_PREPARE.				

last_prober_call

Name	Type	Initial value	First available	Last usable
last_prober_call	**CHAR_P**	**last_prober_call**		
Pointer to the last prober function which encountered an error. This item is updated in KTXESUP/KTXEProberErrorMessage which is used by the Execution Engine. This data pool value can be used at UAP_PRB_ERR_HDLR.				

last_prober_error

Name	Type	Initial value	First available	Last usable
last_prober_error	**INT_P**	**last_prober_error**		
Pointer to the last prober which encountered an error. This item is updated in KTXESUP/KTXEProberErrorMessage which is used by the Execution Engine. This data pool value can be used at UAP_PRB_ERR_HDLR.				

last_subsite

Name	Type	Initial value	First available	Last usable
last_subsite	**LONG_P**	**last_subsite**		
Pointer to the last subsite structure. Updated after subsite change. This is used to assist in getting subsite timing information. Refer to KTXEAddIn:KTXEShowSubsiteTime module for details.				

limit_list

Name	Type	Initial value	First available	Last usable
limit_list	**LONG_P**	**limit_list**	**UAP_VALIDATE_OCR**	**UAP_WAFER_END**
Pointer to the limits list returned by GetLimit. Updated each wafer, if the Wafer Plan file has a limits file specified.				

limithashtab

Name	Type	Initial value	First available	Last usable
limithashtab	**LONG_P**	**&limithashtab[0]**	**UAP_WAFER_BEGIN**	**UAP_WAFER_END**

Pointer to the hash table used for fast lookups of limit values.
Example:

```
LIMIT **limithashtab;
LIMIT *limit_list;

limithashtab = (LIMIT**)dpGetPointer( "limithashtab"' LONG_P);
limit_list = (LIMIT *)limithashtab[ hash( result_id, HASHSIZE ) ];
while(limit_list != NULL)
{
    if (strcomp( result_id, limit_list->id ) == 0 )
        break ; /* we have the limit we are looking for... */

    limit_list = limit_list->nexth ;
}
```

lot

Name	Type	Initial value	First available	Last usable
lot	**LONG_P**	**lot**	**UAP_PROBER_INIT**	**UAP_ENGINE_EXIT**
Pointer to lot structure.				

lotadd

Name	Type	Initial value	First available	Last usable
lotadd	**INT_P**	**&lotadd**	**UAP_PROBER_INIT**	**UAP_WRITE_LOT_INFO**

Flag that determines if PutLot will append data to an exist lot file, or create a new lot file. Values are APPENDLOT or CREATELOT. Must be set before or at UAP_WRITE_LOT_INFO.
Example:

```
int *lotAdd ;
/* set up for append to lot data
 */
lotAdd = ( int *)dpGetPointer( "lotadd", INT_P ) ;
*lotAdd = APPENDLOT ;
```

lotid

Name	Type	Initial value	First available	Last usable
lotid	**CHAR_P**	**lot->id**	**UAP_PROBER_INIT**	**UAP_ENGINE_EXIT**
Lot id string.				

maxErrEvtLines

Name	Type	Initial value	First available	Last usable
maxErrEvtLines	**INT_P**	**maxErrEvtLines**	**UAP_PROG_ARGS**	**UAP_PROG_ARGS**

Pointer to the maxErrEvtLines variable used to determine the maximum buffer size of the Error/Event Message Dialog window. Value is lines to display. Change this value at UAP_PROG_ARGS.
Example:

```
int *maxLines ;
/* get pointer from data pool
 */
maxLines = ( int *)dpGetPointer( "maxErrEvtLines", INT_P ) ;
*maxLines = newMaxLinesValue ;
```

maxScrollLines

Name	Type	Initial value	First available	Last usable
maxScrollLines	**INT_P**	**maxScrollLines**	**UAP_PROG_ARGS**	**UAP_PROG_ARGS**

Pointer to the maxScrollLines variable used to determine the maximum buffer size of the Scrolling Msg dialog window. Value is lines to display. Change this value at UAP_PROG_ARGS.
Example:

```
int *maxLines ;
/* get pointer from data pool
 */
maxLines = ( int *)dpGetPointer( "maxScrollLines", INT_P ) ;
*maxLines = newMaxLinesValue ;
```

next_wwp_list

Name	Type	Initial value	First available	Last usable
next_wwp_list	**LONG_P**	**&previous_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
Pointer to the next WWP list node that is scheduled to be executed. This is updated each time a macro is executed. Structure definition: See wwp_list_t in ktxe_types.h. definition.				

next_wwp_list_ptr

Name	Type	Initial value	First available	Last usable
next_wwp_list_ptr	**LONG_P**	**&previous_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
This is a pointer to the next_wwp_list pointer value. This can be adjusted to alter execution flow if desired.				

previous_wwp_list

Name	Type	Initial value	First available	Last usable
previous_wwp_list	**LONG_P**	**&previous_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
Pointer to the previous WWP list node that was actually executed. This is updated each time a macro is executed. See type definition for wwp_list_t in $KIHOME/include/ktxe_types.h.				

prev_wwp_list_ptr

Name	Type	Initial value	First available	Last usable
prev_wwp_list_ptr	**LONG_P**	**&previous_wwp_list**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
This is a pointer to the previous_wwp_list pointer value. This can be adjusted to alter execution flow if desired.				

prober_wafer_id

Name	Type	Initial value	First available	Last usable
prober_wafer_id	**CHAR_P**	**&prober_wafer_id**	**UAP_VALIDATE_OCR**	**UAP_WAFER_END**

Pointer to the wafer id as read from the prober OCR, if supported. If prober does not support OCR, wafer id is determined by value in cassette plan, or default value of Wafer_xx, where xx is the slot number.

Example:

```
char *stringPtr ;

stringPtr = ( char *)dpGetPointer( "prober_wafer_id", CHAR_P ) ;
printf( "old id: <%s>\n", stringPtr ) ;

/* Make sure you do not over-run the array!!
 * Currently prober_wafer_id is sized 256 characters
 */
strcpy( stringPtr, "A different wafer id" ) ;
```

product_file

Name	Type	Initial value	First available	Last usable
product_file	**CHAR_P**	**&product_file**	**UAP_PROG_ARGS**	**UAP_PROBER_INIT**
Prober Product file string. Must be updated at or before UAP_PROBER_INIT.				

result_list

Name	Type	Initial value	First available	Last usable
result_list	**LONG_P**	**NULL**	**UAP_TEST_END**	**UAP_HANDLE_ABORT**
Pointer to the results list created by a KTM.				

site

Name	Type	Initial value	First available	Last usable
site	**LONG_P**	**site**	**UAP_SITE_CHANGE**	**UAP_SITE_END**
Pointer to the site structure. Updated before `UAP_SITE_CHANGE`.				

sites_tested

Name	Type	Initial value	First available	Last usable
sites_tested	**INT_P**	**&sites_tested**	**UAP_SITE_CHANGE**	**UAP_ENGINE_EXIT**
Pointer to number of sites tested. Updated before UAP_SITE_CHANGE.				

skip_first_wafer_load

Name	Type	Initial value	First available	Last usable
skip_first_wafer_load	**INT**	**skip_first_wafer_load**	**UAP_PROG_ARGS**	**UAP_POST_LOT_INFO**
Flag to skip the first KTXE PrLoad, PrProfile, and PrAlign calls. Must be updated at or before UAP_WRITE_LOT_INFO. Use the kult routine KTXEAddIn:KTXEOperatorLoadAlign routine at UAP_WRITE_LOT_INFO. This flag is used for P8 and other probers that need the first wafer loaded manually.				

skip_next_wafer_load

Name	Type	Initial value	First available	Last usable
skip_next_wafer_load	**INT**	**skip_next_wafer_load**	**UAP_PRB_ERR_HDLR**	**UAP_WAFER_END**
Flag to skip the next KTXE PrLoad, PrProfile, and PrAlign calls. Must be updated at UAP_PRB_ERR_HDLR. Use the kult routine KTXEAddIn:KTXESkipNextWaferLoad routine at UAP_PRB_ERR_HDLR. This flag is used for any prober that the operator can, during testing, cycle to the next wafer manually.				

subsite

Name	Type	Initial value	First available	Last usable
subsite	**LONG_P**	**subsite**	**UAP_SUBSITE_CHANGE**	**UAP_SUBSITE_END**
Pointer to the subsite structure. Updated at subsite change.				

sum_report_options

Name	Type	Initial value	First available	Last usable
sum_report_options	**CHAR_P**	**sum_report_options**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
Summary report options from KTXE "-s" command-line switch.				

total_sites

Name	Type	Initial value	First available	Last usable
total_sites	**INT_P**	**&total_sites**	**UAP_WAFER_BEGIN**	**UAP_ENGINE_EXIT**
Pointer to total number of sites to be tested. Updated before UAP_WAFER_BEGIN.				

total_wafers

Name	Type	Initial value	First available	Last usable
total_wafers	**INT_P**	**&total_wafers**	**UAP_POST_PROBER_INIT**	**UAP_POST_ENGINE_EXIT**
Pointer to total wafers to be tested value. Updated after UAP_PROBER_INIT.				

user_arg

Name	Type	Initial value	First available	Last usable
user_arg	**CHAR_P**	**user_arg**	**UAP_PROG_ARGS**	**UAP_ENGINE_EXIT**
User argument entered from "-u" command-line switch.				

wafer

Name	Type	Initial value	First available	Last usable
wafer	**LONG_P**	**wafer**	**UAP_VALIDATE_OCR**	**UAP_WAFER_END**
Pointer to the wafer structure. Updated at wafer change.				

wafers_tested

Name	Type	Initial value	First available	Last usable
wafers_tested	**INT_P**	**&wafers_tested**	**UAP_WAFER_BEGIN**	**UAP_WAFER_END**
Pointer to number of wafers tested. Updated before UAP_WAFER_BEGIN.				

wdfptr

Name	Type	Initial value	First available	Last usable
wdfptr	**LONG_P**	**wdfptr**	**UAP_POST_PROBER_INT**	**UAP_POST_ENGINE_EXIT**
Pointer to the wdfptr, or wafer description structure. Updated before UAP_POST_PROBER_INIT.				

wpf_info

Name	Type	Initial value	First available	Last usable
wpf_info	**LONG_P**	**wpf_info**	**UAP_VALIDATE_OCR**	**UAP_ENGINE_EXIT**
Pointer to the wafer plan structure. Used by the Analysis Software.				

wwp_list

Name	Type	Initial value	First available	Last usable
wwp_list	**LONG_P**	**wwp_list**	**UAP_WAFER_BEGIN**	**last UAP_SITE_END**
Pointer to the working wafer plan list (WWP list). Updated each wafer, each test, each abort. See type definition for wwp_list_t in $KIHOME/include/ktxe_types.h.				

Advanced data pool use

You can manipulate the data pool in the Keithley User Library Tool (KULT) modules using the data pool functions. You must be careful about using these functions because the improper use may cause fatal errors. The functions in the following list allow you to add, modify, remove, and print data pool information.

To use the data pool functions in KULT, you must include the `COM_usrlib.h` header file. Type `#include "COM_usrlib.h"` in the Include Files section of the KULT Parameters window.

Most of the data pool functions will require you to pass the data type of the variable that you want to manipulate. You must pass in one of the following data types:

- `INT`, `INT_P`: Integer, Integer Pointer
- `FLOAT`, `FLOAT_P`: Float, Float Pointer
- `LONG`, `LONG_P`: Long, Long Pointer
- `DOUBLE`, `DOUBLE_P`: Double, Double Pointer
- `CHAR`: Character
- `CHAR_P`: String of Characters
- `INT_ARRAY`: Array of Integers
- `FLOAT_ARRAY`: Array of Floats
- `DOUBLE_ARRAY`: Array of Doubles

The following is a list of data pool functions with the descriptions below.

Adding items to the data pool:

```
int dpAddData(char *name, int type, ...);
int dpAddPointer(char *name, int type, void *valuep);
int dpAddArray(char *name, int type, void *valuep, int elements);
```

Getting items from the data pool:

```
void      *dpGetDataPtr(char *name, int type);
void      *dpGetPointer(char *name, int type);
void      *dpGetArrayElement(char *arrname, int type, int element);
```

Removing items from the data pool:

```
void      dpRemoveData(char *name, int type);
```

Printing items in the data pool:

```
void      dpPrintData(char *name, int type);
void      dpPrintAllData(void);
```

Data pool function descriptions

The following table contains the data pool commands and a brief description of what they do. For detailed descriptions of these commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
dpAddData	Adds new non-pointer type data to the data pool.
dpAddPointer	Adds pointer-type data to the data pool.
dpAddArray	Adds arrays to the data pool.
*dpGetDataPtr	Gets a pointer to a value of user-defined non-pointer type data (for example, INT, FLOAT, DOUBLE, LONG, CHAR) in the data pool.
*dpGetPointer	Gets the pointer value of user-defined pointer type data (for example, INT_P, FLOAT_P, DOUBLE_P, LONG_P, CHAR_P) in the data pool.
*dpGetArrayElement	Gets a specific element of an array from the data pool.
dpRemoveData	Removes specific data from the data pool.
dpPrintData	Prints a variable and its value in the data pool by passing the name and type of data.
dpPrintAllData	Prints all the data in the data pool.

Example

`dpAddData`/`dpGetDataPtr` example:

```
{
   /* create/initialize value in data pool
    */
   int value ;

   value = 4 ;
   dpAddData( "nameOfElement", INT, value ) ;
}
{
   /* get value from data pool
    */
   int value ;

   value = *(( int *)dpGetDataPointer( "nameOfElement", INT )) ;

   /* value would now equal 4
}
```

`dpAddPointer`/`dpGetPointer` examples:

```
   /* create/initialize value in data pool
    */
   double *dblPtr ;

   /* NOTE: This malloc needs to occur only once. DO NOT call
    * this inside of a loop.
    */
   dblPtr = ( double *) malloc( sizeof( double ) ) ;
   *dblPtr = 5.6 ;

   dpAddPointer( "myPtrName", DOUBLE_P, dblPtr ) ;
}
{
   /* Get value from data pool
    */
   double *dblPtr ;
   double dbl ;

   dlbPtr = ( double *)dpGetPointer( "myPtrName", DOUBLE_P ) ;
   dbl = *dlbPtr ;
   /* dbl = 5.6 now */

   /* Modify value. NOTE that we do not have to "refresh" the
    * data pool. Since this is a pointer data pool value
    */
   *dblPtr = 7.8 ;
}
{
   /* Read modified data pool value
    */
```

```
    double *dblPtr ;
    double dbl ;

    dlbPtr = ( double *)dpGetPointer( "myPtrName", DOUBLE_P ) ;
    dbl = *dlbPtr ;
    /* dbl = 7.8 now */
}

    /* CHAR_P usage:
     */
{
    char *stringVar ;

    /* Create and initialize data pool entry
     */
    stringVar = ( char *)malloc( 128 ) ;

    /* Make sure you do not over-run the array size!!
     */
    strcpy( stringVar, "Message goes here" ) ;

    dpAddPointer( "myString", CHAR_P, stringVar ) ;
}
{
    char *stringPtr ;

    stringPtr = ( char *)dpGetPointer( "myString", CHAR_P ) ;
    /* stringPtr contains "Message goes here" */

    /* Make sure you do not over-run the array size!!
     */
    strcpy( stringPtr, "A different message" ) ;
}
{
    char *stringPtr ;

    stringPtr = ( char *)dpGetPointer( "myString", CHAR_P ) ;
    /* stringPtr contains "A different message" */
}
```

User access points (UAPs)

User access points (UAPs) are supported to let you extend the features and functions of a Keithley provided test execution engine. This is done by writing, compiling, and building user libraries. No Keithley provided code needs to be rebuilt. Multiple test execution engines are supported; however, it is expected that necessary execution engine extensions can be accommodated within predetermined UAPs.

UAPs are defined for all execution engines in the system `ktpm.ini` file. An example follows:

```
/* Start of File */
<Engine_List>
Engine1 = ktxe,$KIBIN,Initial KI Execution Engine for V5.x
<ktxe>
UAP1 = UAP_PROG_ARGS,process user's ktxe command line arguments
UAP2 = UAP_CASSETTE_LOAD,start processing the cassette plan
UAP3 = UAP_LOT_INFO,start setting up the lot file
UAP4 = UAP_PROBER_INIT,start of prober initialization
UAP5 = UAP_WAFER_MISMATCH,mismatch detected between cassette plan and prober
UAP6 = UAP_ACCESS_WDF_INFO,allow access to wdf before call to PrInit
UAP7 = UAP_POST_PROBER_INIT,called to send init commands to prober
UAP8 = UAP_WRITE_LOT_INFO,before writing lot information
UAP9 = UAP_POST_LOT_INFO,write usertag data after LOT header
UAP10 = UAP_WAFERLOAD_STATUS,after a wafer load and the wafer is rejected
UAP11 = UAP_ALIGN_ERROR,error recovery after wafer alignment
UAP12 = UAP_POST_INITIAL_WAFER_LOAD,perform AutoZ after first wafer load
UAP13 = UAP_WAFER_PREPARE,start processing next wafer plan
UAP14 = UAP_VALIDATE_OCR,after OCR and before wafer ID is logged to data file
UAP15 = UAP_WAFER_BEGIN,start executing wafer plan
UAP16 = UAP_SITE_CHANGE,start of next site processing
UAP17 = UAP_SUBSITE_CHANGE,start of next subsite processing
UAP18 = UAP_TEST_BEGIN,start of next ktm processing
UAP19 = UAP_TEST_END,end of processing a ktm
UAP20 = UAP_TEST_DATA_LOG,after test data has been logged
UAP21 = UAP_HANDLE_ABORT,processing an abort condition
UAP22 = UAP_SUBSITE_END,end of current sub-site processing
UAP23 = UAP_SITE_END,end of current site processing
UAP24 = UAP_WAFER_END,end of processing a wafer plan
UAP25 = UAP_LOT_END,end of processing the cassette plan file
UAP26 = UAP_ENGINE_EXIT,before leaving the execution engine
UAP27 = UAP_ABORT_EXIT_HDLR,called as an atexit function
UAP28 = UAP_PRB_ERR_HDLR,called if prober err and function exists
UAP29 = UAP_STATUS_CHANGE,called when pause/cont is pressed on StatDlg
UAP30 = UAP_PROFILE_WAFER,before profiling a wafer
# Place probers that support PrSetSlotStatus here in this list.
# The YES is mandatory!!!
<AbsProbers>
EG40=YES
EG2X=YES
TSK9=YES
P8=YES
# This is used for working off-line. Comment this out if you want.
FAKE=YES
/*End of File */
```

User access point usage

User access points (UAPs) can access data items stored or referenced through the data pool. See Data pool (on page 6-228) for more information.

The following is an explanation of what data is accessed or passed at each UAP:

- `UAP_PROG_ARGS`: At this access point, program arguments passed on the command line are available.
- `UAP_CASSETTE_LOAD`: Cassette plan processing begins at this point.
- `UAP_LOT_INFO`: This access point is located before the call to `lotdlg`; if additional information needs to be added to the Lot Information dialog before the operator sees it, here is a good spot.
- `UAP_PROBER_INIT`: The prober will be communicated with for the first time after this access point. If product files on the prober are used, call `KTXEGetproductFile` to set the product filename.
- `UAP_WAFER_MISMATCH`: This UAP is positioned such that it may be used to detect the following situation: In the event of a failure that orphans wafers in the pipeline. The pipeline is defined as: Quick Loader, Pre-Aligner and Chuck. At the `UAP_WAFER_MISMATCH UAP` the wafer positions can be queried (not all probers allow this action; refer to the documentation for your prober), and a determination can be made either to continue testing or abort the test. This will potentially save loading and unloading time consumed during wafer recovery.
- `UAP_ACCESS_WDF_INFO`: This UAP will allow the user to access wafer data file (WDF) information prior to prober initialization.
- `UAP_POST_PROBER_INIT`: At this access point, any additional prober initialization commands may be sent.
- `UAP_WRITE_LOT_INFO`: At this access point, any additions to the lot file header may be made before the lot file header is written.
- `UAP_POST_LOT_INFO`: This UAP may be used to add tag data to the lot file before the first wafer is tested.
- `UAP_WAFERLOAD_STATUS`: This UAP may be used to notify a shop-floor control system that the wafer load was completed but the wafer was rejected.
- `UAP_ALIGN_ERROR`**:** This UAP is positioned after the wafer is commanded to auto-align (not all probers allow this action, please refer to the documentation for your prober). In the event that the alignment fails, this UAP will allow corrective action to take place.
- `UAP_POST_INITIAL_WAFER_LOAD`: At this access point, `AutoZ` is performed after the first wafer load. Note that `AutoZ` is a function of the SofTouch optional licensed feature for the Keithley Test Environment (KTE). `UAP_WAFER_PREPARE`**:** This is the first access point in the wafer loop. The wafer plan files have not loaded for this wafer at this time. Also, the working wafer plan has not been prepared.

- UAP_VALIDATE_OCR: At this access point, the wafer ID read from the prober can be verified or altered. This is the last UAP point before the wafer ID is written to the Keithley Data File (.kdf) file.
- UAP_WAFER_BEGIN: At this point, the working wafer plan has been prepared, the limits file is loaded into the limits structure, and the probe card file has been loaded. The wafer id has also been read from the prober or generated by the execution engine, and the wafer information has been written to the .kdf data file.
- UAP_SITE_CHANGE: At each site change, this access point is called.
- UAP_SUBSITE_CHANGE: At each subsite change, this access point is called.
- UAP_TEST_BEGIN: This UAP is executed immediately before a Keithley Test Module (KTM) is executed.
- UAP_TEST_END: This UAP is executed immediately after a KTM is executed.
- UAP_TEST_DATA_LOG: At this UAP, test results from the most recent KTM may be logged to another location. Use the "result list" data pool item.
- UAP_HANDLE_ABORT**:** This access point is active if any result generated an abort action. It uses the failed_result_list structure to determine which result aborted.
- UAP_SUBSITE_END: At end of subsite processing, this access point is called.
- UAP_SITE_END: At end of site processing, this access point is called.
- UAP_WAFER_END: At the end of the working wafer plan execution, this access point is available. The wafer has not been unloaded. Also, the KDF data file has not been marked as the end of the wafer.
- UAP_LOT_END: Testing of the lot is now complete, and the KDF data file is closed. For a lot summary report, the KTXESummaryReport routine may be used here.
- UAP_ENGINE_EXIT**:** This is the last access point before the execution engine completes execution.
- UAP_ABORT_EXIT_HDLR: This UAP is called when the execution engine is aborted prior to a test plan completion. The SOLARIS atexit function is used to schedule this routine. The Keithley User Interface (KUI) functions may not be used to display information. Use an external program for graphical user interface (GUI) prompts.
- UAP_PRB_ERR_HDLR: If a prober error occurs, this UAP is used to call user created recovery code.
- UAP_STATUS_CHANGE: This UAP is called when the operator presses the Pause or Continue buttons on the Status Dialog Window, causing an execution state change. This UAP point can be used to notify a shop-floor control system that the tester has been paused.

Types of user access points

There are two different types of user access points (UAPs):

- **User library modules:** Created by the Keithley User Library Tool (KULT)
- **Keithley Test Module (.ktm) files:** Created by the Keithley Interactive Test Tool (KITT)

User library modules

The format for user library modules created by the Keithley User Library Tool (KULT) is:

`UAP_aaaa,user_library_name,module_name(argl,arg2)`

- Input arguments come from the data pool
- Output arguments are written to the data pool

Example:

```
UAP_LOT_END,,status = KTXESummaryReport(lot, sum_report_options)
```

Where:

- `lot`: A pointer to a structure (long *) input
- `sum_report_options`: A char string (char *) input
- `status`: The returned value

The parameters `lot` and `sum_report_options` must be in the data pool before this module is accessed. The status return values are created in the data pool if it does not exist or it is updated if it does exist.

User libraries must be the directory tree defined by the environment variable `KI_KULT_PATH` (typically, this is `/opt/ki/usrlib`).

Keithley Test Module files

The formats for .ktm files created by the Keithley Interactive Test Tool (KITT) are:

`UAP_aaaa,,ktm_name` (note default path is KI_KTXE_KTM directory)

`UAP_aaaa,,/path/ktm_name` (full path used)

`UAP_aaaa,,$env_var/ktm_name` (environment variable used; `.ktm` file extension is optional)

Examples:

```
UAP_WAFER_BEGIN, ,measure_air.ktm
```

```
UAP_WAFER_BEGIN, ,/mydirectory/for/uap/ktms/measure_air.ktm
```

```
UAP_WAFER_BEGIN,,$MYUAPKTMS/measure_air.ktm
```

UAP locations

User access points (UAPs) are defined in three different locations:

- Within a UAP file pointed to `$KI_KTXE_SYSTEM_AP`
- Within a standard UAP file
- Within a cassette plan

The UAPs are executed in the order listed above.

Use of LPTLib functions at UAPs

Linear Parametric Test Library (LPTLib) commands cannot be used at the following user access points (UAPs):

- `UAP_PROG_ARGS`
- `UAP_CASSETTE_LOAD`
- `UAP_LOT_INFO`
- `UAP_ENGINE_EXIT`

LPTLib commands may not be used within user library functions or Keithley Test Modules (KTMs) called at these UAPs.

Distributed user libraries

The following libraries and routines are available for use as reference routines for user access point (UAP) code development.

- **KITTAddIn**: Legacy routines that allow the user to insert or extract data elements into or from arrays. Keithley Test Environment (KTE) version 4.2.2 and later supports array notation within Keithley Interactive Test Tool (KITT) macros, making these routines obsolete.
- **KITTSupport**: `PutUserDataLoggingKTXE` routine to allow the display of run-time results.
- **KI_DEBUG**: Debug routines to allow display of datapool and KTE prober structure contents. Also a routine is provided that creates a global data file (`.gdf`) file from the current contents of the data pool. This `.gdf` file can be used by KITT to aid in debugging macros.
- **KI_UAPLIB**: Contains routines that create and use a subset of the limits list. These routines can be used to validate results against the limits list. These routines can be modified to perform any custom processing of results that exceed a limit value. In this library, some Keithley User Interface (KUI) functions are included.
- **KTXEAddIn**: Routines that demonstrate access to data pool values to control Keithley Test Execution Engine (KTXE) execution flow. The `KTXEOperatorLoadAlign` routine, for example, can be used to prompt the operator to load the first wafer manually for P8-type probers.

Please note that these routines are part of the KTE distribution and are subject to change in future releases. It is recommended that you copy and rename the routine you want in your own libraries for use within KITT and KTXE.

Test macro debugging

Since the data pool is only available while executing the Keithley Test Execution Engine (KTXE – Keithley Interactive Test Tool (KITT) does not generate a global data pool), debugging your Keithley Test Module (KTM) in KITT is very difficult. `DBG_gdfCreate` helps solve this problem by creating a snapshot of the current global data pool while executing KTXE.

By executing the `DBG_gdfCreate` macro at a user access point (UAP) just prior to the point your DTM is experiencing problems, it is possible to capture the contents of the global data pool into a `.gdf` file. This `.gdf` file can then be loaded into KITT along with your KTM to simulate the conditions of your KTM running in KTXE.

Following is a technical description of `DBG_gdfCreate` ...

```
(void)DBG_gdfCreate(char *gdffile)
```

`DBG_gdfCreate` is a KITT macro-debugging routine that generates a valid Global Data File (`.gdf`) based on the current contents of the global data pool. This Global Data File will be stored in the location defined by the argument `gdffile`.

`DBG_gdfCreate` may be executed at any user access point to generate a snapshot of the current global data pool contents. This `.gdf` file may then be used in KITT as a means of debugging macros relying on data generated by KTXE at run time.

NOTE

The `KI_KTXE_DEBUG_LOG` environment variable must be set to a valid filename prior to calling `DBG_gdfCreate`. An example of this would be `setenv KI_KTXE_DEBUG_LOG /tmp/log`. `KI_KTXE_DEBUG_LOG` must not be set to `/dev/tty or /dev/null` if this log file is used as temporary storage for the data pool data during the creation of the `.gdf` file.

This function can be found in the `KI_DEBUG` KULT library.

Section 7

Recipe Manager

In this section:

Overview

Keithley Recipe Manager (KRM) is part of the Keithley Test Environment (KTE) software installed on your system. You can use KRM to create, revise, version-control, and process parametric test recipes that gather data for use in semiconductor wafer quality control.

Version control in KRM allows you to manage a large number of recipes by maintaining records and archives of the recipes run on each production lot. The ability to trace the history of a recipe and its supporting files (including input conditions and test algorithms) makes it easier to troubleshoot processes and electrical testing.

A recipe is a version-controlled collection of parameters, files, and instructions that is saved in a version control system for use in production test.

You can use KRM to:

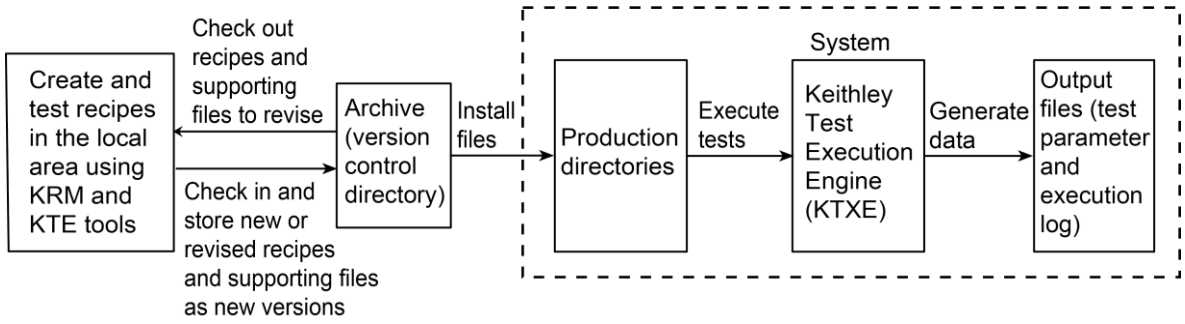
- Develop a new recipe
- Modify an existing recipe
- Modify a common supporting file used across several recipes
- Bundle files for production
- Act as a control point for releasing recipes and individual files to production.

Recipes contain all the information needed to test a specific product, including:

- Test structures on which tests should be run
- Location of the test structures on the wafer
- Tests to run on the test structures
- Adaptive testing rules
- Test result control bands that define when to bin the results
- Operator instructions
- Automation requirements

The following figure is a simplified diagram of the Keithley Recipe Manager (KRM) process.

Figure 99: Engineering, version control, and production control process



Additional features

Some additional features of Keithley Recipe Manager (KRM) include:

- Provides a single point of reference for viewing, editing, and executing recipes
- Organizes recipes in a directory-style structure with folders for products with similar test requirements (called "processes") and subfolders for specific product models (product families)
- Version-controls supporting files and recipes and distributes them to multiple systems
- Controls changes to recipes and the release of recipes into production, providing traceability of changes and test results
- Automates packaging and transfer of recipes to other systems
- Generates documentation and diagnostic reports for a recipe showing the content and characteristics of each recipe

Setting up Keithley Recipe Manager

Keithley Recipe Manager (KRM) is part of the Keithley Test Environment (KTE) software that comes installed on your system.

NOTE

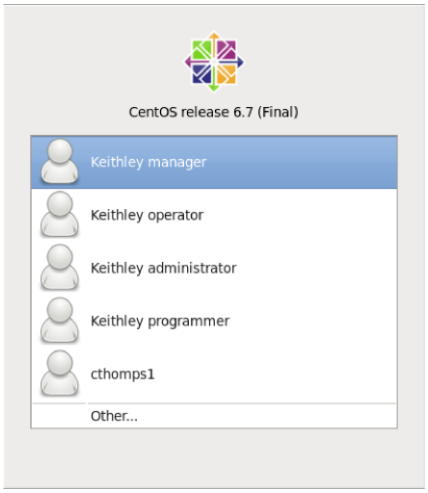
If this is the first time you have used KRM, you will need to configure it for your system. See the following topics for instructions.

System and software passwords

You need to use passwords when you access Keithley Test Environment (KTE) software through the Linux® environment and when performing some operations in the KTE and Keithley Recipe Manager (KRM) software.

When you start your system, a log-in screen for different users is displayed on the system computer.

Figure 100: KTE log in screen



The default passwords for the system log-in screen are:

- Keithley manager: `kthmgr`
- Keithley operator: `kthopr`
- Keithley administrator: `kthadm`
- Keithley programmer: `kthprg`

You can set up individual accounts for users to access the system with their own passwords. For more information about how to do this, see Setting up separate user accounts (on page 7-14).

There are several operations in KTE and KRM that require special release privileges to complete. For example, when you release and install a recipe file or other KTE files, you are prompted for a password. A release password is also needed when doing certain version-control tasks. The default release password is `kthadm`.

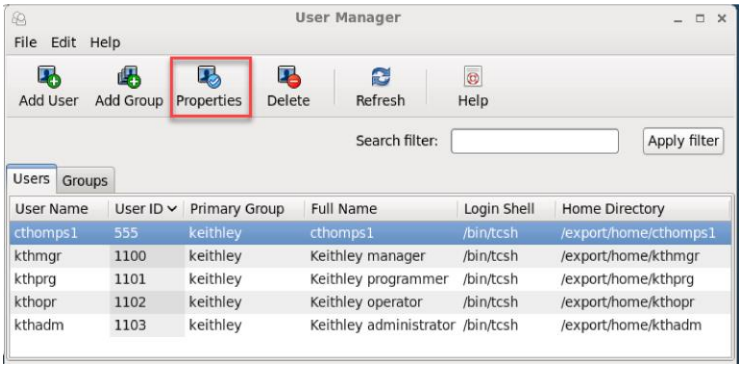
You can set up an administrative group login that allows one or more people with special release privileges to run password-protected operations without having to enter a release password. For more information, see Setting up an administrative group maintenance account (on page 7-14).

Changing passwords

To change a user password:

1. In the main Linux® environment, go to **System > Administration > Users and Groups**. The User Manager interface is displayed.
2. Select the name of the user whose password you need to change and select **Properties**.

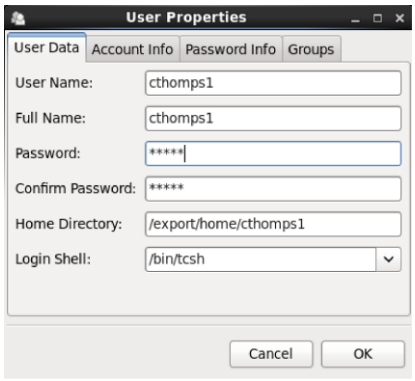
Figure 101: Linux® User Manager interface



The User Properties dialog box is displayed.

3. Enter a new password in the **Password** box.

Figure 102: Enter a new password



4. Enter the password again in the **Confirm Password** box and select **OK**.

NOTE

To change the release password, change the password for the kthadm account. The password assigned to this account is the one you will need when prompted for passwords in the Keithley Test Environment (KTE) tools and in Keithley Recipe Manager (KRM).

System configurations for Keithley Recipe Manager

There are several ways you can use Keithley Recipe Manager (KRM):

- Single-system setup (on page 7-8). The KRM archive is set up on the system and recipes in the archive are available to that system only.
- Multiple networked systems setup (on page 7-10). The KRM archive is set up on one of the systems in the network or on an external computer, and recipes are accessible to all systems in the network.
- Development-only setup (on page 7-13). A development-only system is set up for development of new recipes only.

Preparing information for Keithley Recipe Manager set up

You set up Keithley Recipe Manger (KRM) using command-line utilities (scripts) that prompt you for information. Before starting the scripts, collect the necessary information listed in the following tables (as appropriate for your setup).

Single-system checklist

Server hostname	
System QMO number (serial number)	
Operator account name for system (`kthopr` is default)	
Archive directory location[1] for system (<path/name>)	
Staging directory location[2] for system (<path/name>)	

[1] Record a value if you want a specific name or path-name combination. You can accept the default archive directory location, which is $KIHOME/arch.

[2] Record a value if you want a specific name or path-name combination. You can accept the default staging directory location ($KIHOME/StageQNNNN, where NNNN is the system QMO (serial) number).

Multiple networked systems checklist

Server system	
Server hostname	
System QMO number (serial number)	
System operator account name (`kthopr` is default)	
Archive directory location[3] (<path/name>)	
Staging directory location[4] (<path/name>)	

Client system 1	
Client hostname	
System QMO number[5] (serial number)	
System operator account name (`kthopr` is the default)	
Archive root mount point on this system (<path/name>)	
Staging directory mount point on this system (<path/name>)	
Server hostname	

[3] Record a value if you want a specific name or path-name combination. You can accept the default archive directory location, which is $KIHOME/arch.

[4] Record a value if you want a specific name or path-name combination. You can accept the default staging directory location ($KIHOME/StageQNNNN, where NNNN is the system QMO (serial) number).

[5] For a client, record the client QMO (serial) number. For a development-only system, record DevOnly; the installation script assigns DevelopmentOnly as the QMO number for each development-only system.

Client system 2	
Client hostname	
System QMO number (serial number)	
System operator account name (`kthopr` is default)	
Archive root mount point on this system (<path/name>)	
Staging directory mount point on this system (<path/name>)	
Server hostname	

Client system 3	
Client hostname	
System QMO number (serial number)	
System operator account name (`kthopr` is default)	
Archive root mount point on this system (<path/name>)	
Staging directory mount point on this system (<path/name>)	
Server hostname	

Client system 4	
Client hostname	
System QMO number (serial number)	
System operator account name (`kthopr` is default)	
Archive root mount point on this system (<path/name>)	
Staging directory mount point on this system (<path/name>)	
Server hostname	

Single-system setup

To set up a single system with no connections to other systems, you set up both the server and client on the same system. First you run the server installation script (`$KIBIN/vc_server_install`), then you run the client installation script (`$KIBIN/vc_client_install`). The scripts prompt you for the information you entered in the Single-system checklist (on page 7-5). When you run the client script, use the same information that you used for the server script.

To set up Keithley Recipe Manager on a single system:

1. Log in to the Linux® environment as Keithley manager.
2. Open a terminal window and enable root user privileges by entering the `su` command, followed by the required password. The default password is `keithley`.
3. Run the following script from the command line:

   `$KIBIN/vc_server_install`
4. Enter the information requested by the prompts using the information you entered in the Single-system checklist (on page 7-5).

   The script executes the following sequence of information prompts for the server:

   - Account name (confirm or enter different name)
   - Group ID (confirm or enter different number)
   - Archive directory path (confirm or enter a different path)
   - Staging directory (confirm or enter a different path)
   - Create tester (client) staging area (answer `yes` to set up the client on the same system)
   - Development-only workstation? (`yes` or `no`)
   - Client QMO number (enter number without the Q); for a development-only system, the default QMO number is `DevelopmentOnly` (the next several prompts are skipped for development-only systems)
   - Hostname for the client (confirm or enter a different name)

- Staging directory name for the client (confirm or enter a different name)
- Create another tester (client) staging area (answer `no` for a single-system setup).

The script closes and returns you to the command prompt.

5. Reboot the system.
6. Run the following script from the command line:

   `$KIBIN/vc_client_install`

7. Enter the information requested by the prompts using the same information you entered in the Single-system checklist (on page 7-5).

   The script prompts you for the following information for the client (which is the same as the information for the server):

   - Development-only workstation? (`yes` or `no`)
   - Server computer the same as this client computer? (answer `yes` for a single-system setup)
   - The `/opt/kiS530/bin/vc_server_install` script has been run on the server system computer? (`yes` or `no`; if you started from the beginning of this instruction, answer `yes`)
   - Have you executed the `/opt/kiS530/bin/vc_add_tester_2_server` script or configured the tester's (client) staging area during the server installation process? (`yes` or `no`; if you started from the beginning of this instruction, answer `yes`)
   - Account names (confirm or enter different names)
   - Group ID (confirm or enter different number)
   - Client QMO number (the same number as the server)
   - Server archive root directory (enter with `ServerName: /` preceding the path)
   - Archive root directory (confirm or enter a different directory)
   - Client stage directory (confirm or enter a different directory)

   The script creates the `$KIHOME/.ki_setup_option_5_vc file`, which is subsequently sourced at log-in time and defines the following:

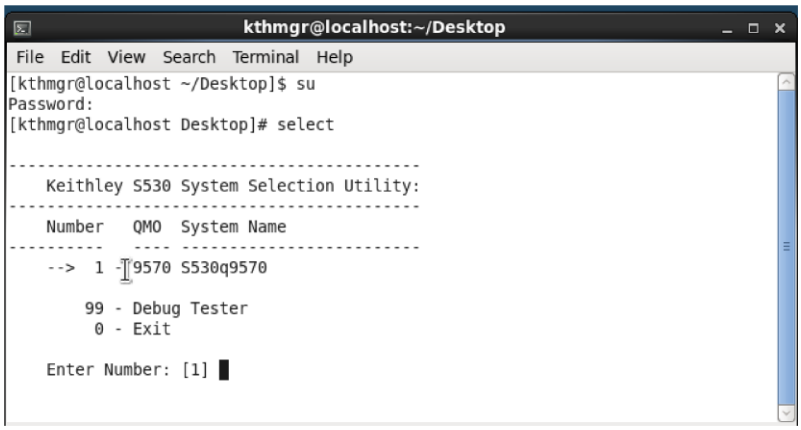
   - The `KI_ARCHIVE` environment variable
   - The `KI_KRM_TESTERS` environment variable, so that it points to the `$KI_ARCHIVE/krm_testers.ini` file
   - The `KI_STAGING_DIR` environment variable if this system is not a development-only workstation
   - The `KI_VC_BIN` environment variable

   The script then configures Keithley Recipe Manager (KRM) so that it can be launched in operator mode from the desktop by right-clicking, and in the shortcut menu that opens, selecting **Scripts > KRM**.

   - Operator account name for this client (default: `kthopr)`
   - Staging directory on the server for this client

8. If you want to populate the archive with existing files, open a new terminal and run the `/opt/kiS530/bin/setuparc` script from the command line. Syntax for the script is `setuparc -f <cpf | all >`, where `cpf` checks in KTE files used by existing cassette plans only, and `all` checks in all available KTE files.
9. At the command prompt, enter `select` to initialize client environment variables. A dialog box similar to the following figure is displayed in the terminal.

Figure 103: KRM system selection utility



10. Enter `1` to configure the connected client.

Multiple networked systems setup

For this configuration, you must set up the server before setting up the client systems.

Server system setup

To set up Keithley Recipe Manager on the server:

1. Log in to the Linux® environment as Keithley manager.
2. Open a terminal window and enable root user privileges by entering the `su` command, followed by the required password. The default password is `keithley`.
3. Run the following script from the command line:

   `$KIBIN/vc_server_install`
4. Enter the information requested by the prompts using the information you entered in the Multiple networked systems checklist (on page 7-6).

   The script executes the following sequence of information prompts for the server:

   - Account name (confirm or enter different name)
   - Group ID (confirm or enter different number)
   - Archive directory path (confirm or enter a different path)
   - Staging directory (confirm or enter a different path)

The script continues, prompting you for information about the clients in the network.

- Create tester (client) staging area (answer `yes` for a multiple networked systems configuration)
- Development-only workstation? (`yes` or `no`)
- Client QMO number (enter number without the Q); for a development-only system, the default QMO number is `DevelopmentOnly` (the next several prompts are skipped for development-only systems)
- Hostname for the client (confirm or enter a different name)

NOTE

The `/etc/hosts` file on the server must be manually populated with the hostnames and IP addresses of all of the client systems that will be mounting to the directories on the server.

- Staging directory name for the client (confirm or enter a different name)
- Create another tester (client) staging area (answer `yes` to add another networked system); the script repeats the tester-specific questions for the new system.
- Create another tester staging area (answer `yes` to keep adding additional systems, or `no` to finish the script).

5. Reboot the system.

Once you have set up the server, you can set up each of the clients using the procedure in Client system setup (on page 7-11).

Client system setup

For the multiple networked systems configuration, complete the following procedure on each client, including a client that shares the same computer as the server.

NOTE

You must set up the server before setting up the clients. See the instructions in Server system setup (on page 7-10).

To set up Keithley Recipe Manager on a networked client:

1. Log in to the Linux® environment as Keithley manager.
2. Open a terminal window and enable root user privileges by entering the `su` command, followed by the required password. The default password is `keithley`.
3. Run the following script from the command line:

   ```
   $KIBIN/vc_client_install
   ```

4. Enter the information requested by the prompts using the information you entered for the first client in the Multiple networked systems checklist (on page 7-6).

The script prompts you for the following information for the client:

- Development-only workstation? (`yes` or `no`)
- Server computer the same as this client computer? (answer `no` for networked client systems)
- The `/opt/kiS530/bin/vc_server_install` script has been run on the server system computer? (`yes` or `no`; if no, complete the instructions in Server system setup (on page 7-10) on the server before continuing)
- Tester (client) staging area configured during the server setup? (`yes` or `no`; if no, run the `/opt/kiS530/bin/vc_add_tester_2_server` script on the server system computer before continuing)
- Account names (confirm or enter different names)
- Group ID (confirm or enter different number)
- Client QMO number
- Server archive root directory (enter with `ServerName: /` preceding the path)

NOTE

The hostname and IP address of the server must be manually populated in the `/etc/hosts` file of each client system.

- Archive mount point on this client; mounts the archive directory from the server and adds an entry to `/etc/fstab`
- Staging directory mount point on the client computer (confirm or enter new path)
- Operator account name for this client (default: `kthopr)`
- Staging directory on the server for this client

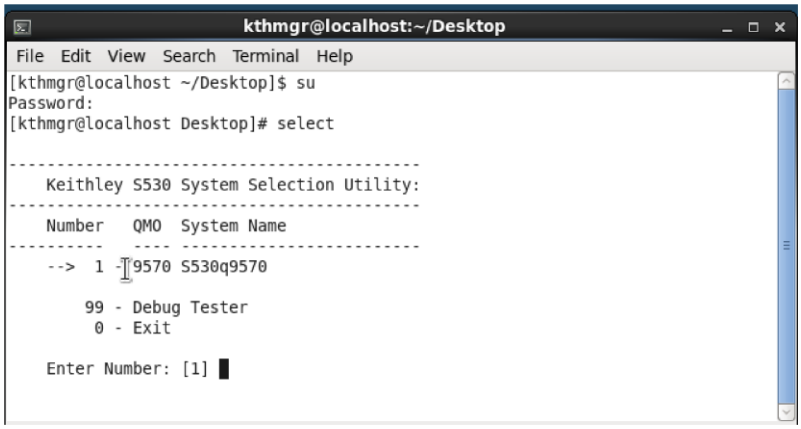
The script creates the `$KIHOME/.ki_setup_option_5_vc` file, which is subsequently sourced at log-in time and defines the following:

- The `KI_ARCHIVE` environment variable
- The `KI_KRM_TESTERS` environment variable, so that it points to the `$KI_ARCHIVE/krm_testers.ini` file
- The `KI_STAGING_DIR` environment variable if this system is not a development-only workstation
- The `KI_VC_BIN` environment variable

The script then configures Keithley Recipe Manager (KRM) so that it can be launched in operator mode from the log in menu on the desktop and puts the KRM icon in the Keithley Tool Palette (KTP).

5. If the server and the client do not share the same computer, reboot the system.
6. At the command prompt, enter `select` to initialize client environment variables. A dialog box similar to the following figure is displayed in the terminal.

Figure 104: KRM system selection utility



7. Enter `1` to configure the connected client.

Development-only system setup

To set up a development-only system:

1. Log in to the Linux® environment as Keithley manager.
2. Open a terminal window and enable root user privileges by entering the `su` command, followed by the required password. The default password is `keithley`.
3. Run the following script from the command line:

   `$KIBIN/vc_server_install`
4. Enter the information requested by the prompts using the information you entered in the Single-system checklist (on page 7-5).
5. The script executes the following sequence of information prompts:
   - Account name (confirm or enter different name)
   - Group ID (confirm or enter different number)
   - Archive directory path (confirm or enter a different path)
   - Staging directory (confirm or enter a different path)
   - Development-only system (`yes` for development-only system)
   - Hostname (confirm or enter different name)
6. When prompted to create another staging area, enter `no`.
7. Reboot the system.

Add KTE files from the local area to the KRM archive

If you have existing supporting files (for example, `.cpf`, `.wdf`, `.klf`, or `.ktm`) that you want to add to the Keithley Recipe Manager (KRM) archive after setting up KRM, check them in using the following procedure.

To check in files from the local area:

1. Log in to the Linux® environment as Keithley manager.
2. At the command prompt, run the following script:

   ```
   setuparc -f [all|cpf]
   ```

   Where:

   - `all` = Check-in of all files that are presently in the local area
   - `cpf` = Check-in only files that are used by the cassette plan files

Setting up separate user accounts

The Keithley Recipe Manager (KRM) engineering mode allows developers to work under individual accounts in individual local areas.

To set up an individual account and local area, send `$KIBIN/create_user_account` from the command prompt. This script prompts you for a user name, group ID (must be the same as the Keithley manager account), account home directory location, and user password. After you confirm the information, the script asks you if you want to create a local working project.

Setting up an administrative group maintenance account

You can set up a Linux® group account that allows one or more users with Keithley Recipe Manager (KRM) release privileges to release and install files without entering a password.

To set up an administrative group maintenance account:

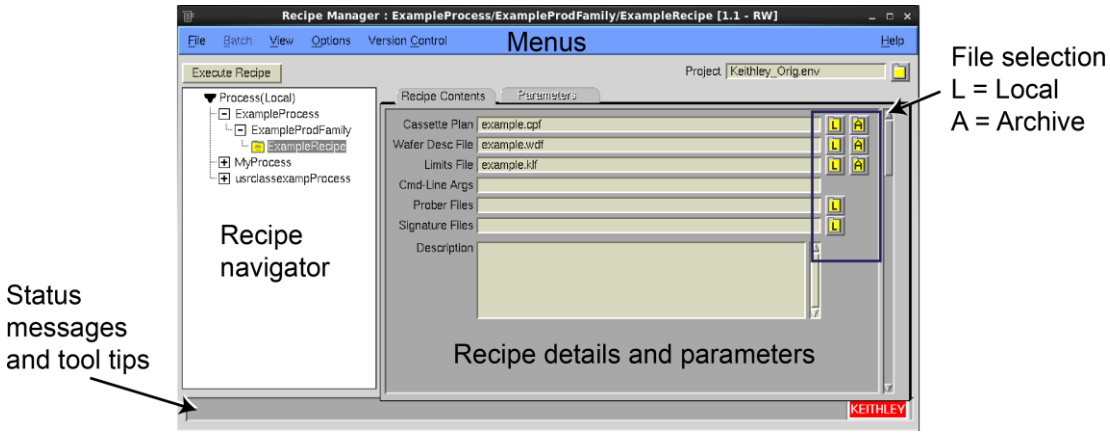
1. In the main Linux® environment, go to **System > Administration > Users and Groups**.
2. Create a new group named `krmadm`.
3. Add each user that you want to have release and install privileges to the group.

The Keithley Recipe Manager user interface

A recipe is single point of reference for all of the Keithley Test Environment (KTE) files and parameters needed to test a specific product. Keithley Recipe Manager (KRM) allows you to create, edit, and version-control recipes for use in a production test environment.

The KRM user interface is shown in the following figure.

Figure 105: The Keithley Recipe Manager user interface



Recipe navigator: On the left side of the interface, the recipe navigator displays recipes organized in a directory structure under the process (a category of products with similar testing requirements) and product family (product model) that they are used with.

Recipe details and parameters: The right side of the interface contains the Recipe Contents and Parameters tabs. The primary supporting files and parameters specified on these tabs define the recipe. The yellow folders in the interface represent file selections in the local and archive areas.

Menus: The menus at the top of the interface allow you to create, revise, version-control, and process recipe files for use in a production environment. Descriptions of the menu options follow this topic.

Execute Recipe button: If a recipe is selected in the recipe navigator, selecting this button executes the recipe.

Project: In the upper right area of the interface, the project in KRM is a predetermined directory structure that organizes files for test recipe development in the local area. The project, which has a `.env` file extension, is defined by a collection of environment variables in the local area of the system. For more information about KRM projects, see Creating projects (on page 7-24).

Status messages and tool tips area: The bottom of the interface displays status information, and when you hover over an item in the interface, shows information about it.

File menu

The options on the File menu are described in the following table.

Menu item	Description
New Process	Create a new process, product family, and reference recipe in the local area (see Creating a new process and product family (on page 7-40)).
New Product Family	Create a new product family and reference recipe under an existing process (see Creating a new product family under an existing process (on page 7-41)).
Save	Save the recipe (see Saving a recipe).
Save As	Save the recipe with a different file name (in the same directory)
New Recipe	Create a new recipe under an existing process and product family; if a reference recipe exists for the product family, the reference recipe is copied and saved under a name you specify for the new recipe; if no reference recipe exists, an empty recipe is created (see Creating a new recipe (on page 7-42)).
Rename Recipe	Rename an existing recipe (see Renaming a recipe (on page 7-50)).
Delete	Delete the selected recipe (see Deleting items from the local area (on page 7-64)).
Exit	Exit Keithley Recipe Manager (KRM)

Batch menu

The Batch menu allows you to perform certain common operations to multiple marked product families or recipes at the same time. Batch operations are only available in the local view.

The options in the Batch menu are described in the following table.

Menu item	Description
Save Copies To	Copies the selected recipe to all marked families, and updates the Name and Time fields. If the selected recipe was archived, then it archives the new recipes. If the head label of the selected recipe was PROD, the new recipes are assigned a PROD label and are added to the Install Request log. If a recipe with the same name already exists in any of the marked families in the local area or archive, no copy is made.
Search/Replace	If recipes are marked and an item on the Recipe Contents tab is highlighted, the contents of the highlighted edit field are written to all marked recipes.
Search/Replace/Release	If recipes are marked and an item on the Recipe Contents tab is highlighted, the contents of the highlighted edit field are written to all marked recipes. As required, the recipes may be checked out, altered, check back in, and released. This applies a PROD label to the recipes and adds them to the install request queue. A release password is required for this operation.
Check Out	Checks out all marked recipes and locks them in the archive.
Check In	Checks in all marked recipes to the archive with a common log comment. Files are checked in to the latest revision with no labels.
Label	Labels the head version of all marked files that have been archived with the specified label. If there is a labeling conflict, the existing label is moved to the head revision.
Remove Label	Removes the labels of all marked files with a specified label. All marked files must be unlocked in the archive (not checked out).

Menu item	Description
Release	Labels all marked files with a PROD label and adds them to the `InstallReq` log. A release password is required for this operation.
UnRelease	Removes the PROD label from all marked files and adds a DELETE flag in the InstallReq log. A release password is required for this operation.
Delete	Deletes all marked recipes from the local directory. If the recipes are not archived, this operation is not reversible. A release password is required for this operation.
UnMark All	Unmarks all marked nodes, including hidden nodes, in a single operation.

View menu

The options on the View menu are described in the following table.

Menu item	Description
Parameter List	Toggles the view of the Parameters tab on and off; the Parameters tab displays the name of each parameter, test macro, and user module in which each parameter is used. You can access the Keithley Interactive Test Tool (KITT), Test Structure Editor (TSE), and Keithley User Library Tool (KULT) by right-clicking the cells under KTM Name, Module Name, and Library Name on the Parameters tab.
Local View	Displays the recipes that are in the local directory; recipes in the local directory can be revised and later moved back to the archive directory.
Archive View	Displays the latest (Head) version of a recipe stored in the archive directory. A recipe in the Archive View is read-only, but you can view and modify a copy in the local area by selecting **Version Control > Other Options > Fetch**. You can use this method to populate a new local area. The Recipe Contents tab is not accessible in this view.
CopyOfProd View	Displays the recipes and primary supporting files in the `CopyOfProd` directory, which contains a complete, unbundled set of the presently-installed, production-labeled (PROD) recipes and supporting files. A recipe in this view is read-only, but the Recipe Contents tab is accessible for viewing.
Refresh Recipe List	Refreshes the recipe navigator display. This is useful in the Keithley Recipe Manager (KRM) operator mode when recipes have been updated in the local production area after transfer from the staging directory of the system.
View Install Request Log	Displays the entries in the `InstallRequest.log` file. This view shows the names of edited files that have not been released and installed (`.krf`, `.cpf`, `.klf`, `.ktm`, and other supporting files).
View Install Record Log	Displays the entries in the archive `InstallRequest.log` file. This file contains a history of each install operation, including installation pass-fail status, the file bundle name, the name of the staging directory where the file bundle was installed, and the date and time of the installation.

Options menu

The selections on the Options menu are described in the following table.

Menu item	Description
View/Set Recipe Contents	Displays all files in the presently selected recipe with revision labels. This allows you to view PROD-labeled versions of the recipe or specify non-PROD versions of the files to create a specific recipe (a recipe for use in the local area, typically for diagnostic purposes). Double-click a revision to enter a new revision number and select **Update** to create a specific recipe.
Display Recipe Contents	Displays a read-only, text-based list of PROD-labeled files and file versions in the presently selected recipe. The listed files are the same files displayed in the View/Set Recipe Contents option. You can save and print the file list from the Display Recipe Contents view.
Load Recipe	Loads the latest version of the recipe and its supporting files from the archive to the local directory. Take care not to overwrite checked out files in the local area.
Unload Recipe	Deletes supporting KTE and user files in the local area. Take care that the files you specify in this operation are not used in other recipes; deleting files using this menu option makes the files unavailable to other recipes until you reload the files by selecting **Options > Load**.
Validate Recipe	Using the Keithley Test Documentation Tool (KTDT), checks the usability of a recipe without hardware execution. Verifies syntax in recipe elements, presence of referenced files (and their versions) in specified locations, availability of user access point (UAP) code, and the presence and validity of parameters. When prompted to load recipe files before validating, do not select Yes unless you are sure that the checked-out supporting files in the local area may be safely overwritten. Select **Validate** and **Display** (and any other details you want to view), and then select **Create Report** to view the results.
Edit krm_tester.ini File	Use this option on servers in a multiple networked systems configuration to add or remove clients from the `krm_testers.ini` file.

Version Control menu

The options in the Keithley Recipe Manager (KRM) Version Control menu are described in the following table.

NOTE

Version control options are available in most of the Keithley Test Environment (KTE) tools. The Version Control menus are similar but have some differences. The Keithley Recipe Manager (KRM) Version Control menu is described below; for more information about the Version Control menu in KTE tools, see Version control (on page 8-1).

Menu item	Description
Show Status	Displays the file name and directory path, file lock status, read/write permissions, current revision number in the local area, most recent (head) version in the archive, and the PROD-labeled version in the archive.
Modify	Combines the Version Control > Other Operations > Check Out operation (automatically selects the PROD-labeled version) and the Options > Load Recipe operation (the Load Recipe option is not available in the Options menu of other KTE tools and is not included in the Modify operation of other KTE tools). You are prompted whether to overwrite each of the recipe files or cancel the load recipe operation.
Deliver	Combines the Options > Load Recipe and Validate Recipe operation, Version Control > Check For Locks operation, Version Control > Other Operations > Check In and Release operations, and Version Control > Install operation. The PROD-labeled version of the recipe is automatically selected. Make sure that checked out supporting files in your local area may be safely discarded before selecting **Yes** at the Load recipe before validating prompt.
Other Operations	**Fetch:** Places a read-only copy of the selected archive file in the local area without locking it. **Check Out:** Copies the selected file from the archive to the local area, places a lock on the file in the archive, and sets write permissions to the version in the local area to allow editing of the file. **Revert:** Removes the lock from a checked-out file and discards any changes that were made while it was checked out, restoring the file to the head (latest version before file was checked out) revision. **Check In:** Places an updated copy of a file in the archive; the revised file is assigned a new revision number and becomes the latest (head) version. If you select **Set Revision** in the Check In dialog box, you can specify a revision offset number, which allows you to have identical files for different models in the archive (the revision offset differentiates the files). **Release:** Release labels a file as production-ready and enters the file name in the Install Request file. The Release operation combines the Options > Load Recipe operation, Validate Recipe operation, Version Control > Check For Locks operation, Select Version operation, and Version Control > Install operation. Make sure that checked out supporting files in your local area may be safely discarded before selecting Yes at the Load recipe before validating? prompt. **Remove Label:** Reverses the Release operation before installing the file in system staging directories, removes the PROD label from the file, and removes the file name from the `InstallRequest.log` file.
History	Displays the revision history and lock status of a file, and identify which versions presently have a PROD label and head (latest file) designation.
Compare	Compares one version of a file with another archived version of the file. When comparing a recipe, the operation displays a list of related files that changed between versions. For more information, see Comparing versions of a file (on page 8-22).
Where used	This menu selection is not useful in Keithley Recipe Manager, but in other KTE tools it allows you to see a list of files that use the presently selected file. See Viewing where a supporting file is used (on page 8-23).
Recipe.adm Operations	Available only when a recipe is selected and a `<recipename>.adm` file exists. Allows you to query the history of the `<recipename>.adm` file, compare revisions of the `<recipename>.adm` file, and view the contents of the `<recipename>.adm` file.
Check for Locks	In the Keithley Recipe manager tool only, checks whether any of the files related to the selected recipe are locked in the archive, identifying supporting file versions that may be in the process of revision.

Menu item	Description
Install	Bundles each newly released PROD recipe file with copies of all its supporting files into a single bundle (`.tar` file). In other KTE tools, bundles newly released PROD supporting files. It then sends a copy of the bundle to the staging directory (or directories in the networked systems configuration) for subsequent transfer by the system. The Install operation deletes the names of the recipe file and its supporting files from the Install Request file.

The primary version control areas

Before developing recipes, you should understand the primary areas of a Keithley Recipe Manager (KRM) version control system. The following topics describe these areas.

Local area

A local area is made up of a series of directory locations that are used to hold recipe files and supporting files during development.

A Keithley Recipe Manager version-control system needs only one local area, a default version of which is created when KTE is installed. However, several may be created and used to meet specific needs of multiple developers. Each developer can have one or more local areas. A local area is created when you create a project using the `make_project` command-line utility script. For more information about creating a project, see Creating a project (on page 7-25).

You select a local area by either using the KRM interface or by executing the `select_project` command-line utility script. For more information about selecting a project, see Selecting a project (on page 7-29).

Archive

The archive, sometimes called the revision control directory, is a directory that contains the following:

- Master copies of all of the recipe and supporting files in the Keithley Recipe Manager version-control system
- Version-numbered change files that define all revisions to all recipe and supporting files
- `InstallRequest.log` and `InstallRecord.log` files

The archive is a permanent repository for every file. All recipe and file development and production testing is done using copies of files from the archive. If a changed file is checked in to the archive, the changes to the file are stored in the archive with a new revision number.

There is only one archive in the system, which may be used by any number of client systems.

Staging directories

A staging directory is a directory that temporarily stores installed recipes and supporting files as file bundles (`.tar` files). A recipe bundle stores a complete, production-ready normal or specific recipe and all of its supporting files. Other types of bundles contain separate Keithley Test Environment (KTE) files or user libraries. There is a staging directory for each client system, and all staging directories contain identical bundles.

Staging provides the following benefits:

- Allows developers to install new and updated files into temporary locations, at any time, without disrupting in-process client system operations.
- Allows the client systems to retrieve new and updated files when they are ready.

It is helpful to set up at least one extra staging directory that is unassociated with a client system. The extra staging directories can be used to transfer recipe bundles to remote sites or to other areas for archiving or backup. Set up such a staging directory in `krm_testers.ini` by setting up a nonexistent client system (refer to Editing the krm_testers.ini file (on page 8-13)).

Normal and specific recipes

In Keithley Recipe Manager (KRM) there are the following types of recipes:

- **Normal recipe:** A normal recipe calls only PROD labeled versions of supporting files.
- **Specific recipe:** A specific recipe calls at least one non-PROD labeled version of a supporting file, commonly for diagnostic purposes. For example, if after updating a supporting file, recipe test results are unusual, you can create a specific recipe that calls the previous version of the supporting file, then rerun the test and compare the results.

To create a specific recipe, start with a normal recipe and reset the revision labels of one or more supporting files. Refer to Creating a specific recipe (on page 7-45) for more information.

Specific recipes are distributed, stored, and run with special processing. Refer to Client systems (on page 8-27) for information.

Getting started

The following topics describe starting KRM and give an overview of the user interface.

Starting Keithley Recipe Manager

You can start Keithley Recipe Manager (KRM) in one of the following modes:

- **Operator mode.** This mode restricts available functions to selecting and executing recipes.
- **Engineering mode.** This mode allows recipe development, version control, validation, and moving files to and from archives and production.

Starting in operator mode

The operator mode in Keithley Recipe Manager (KRM) limits the menu selections to only those needed for an operator's workflow (typically recipe selection and execution).

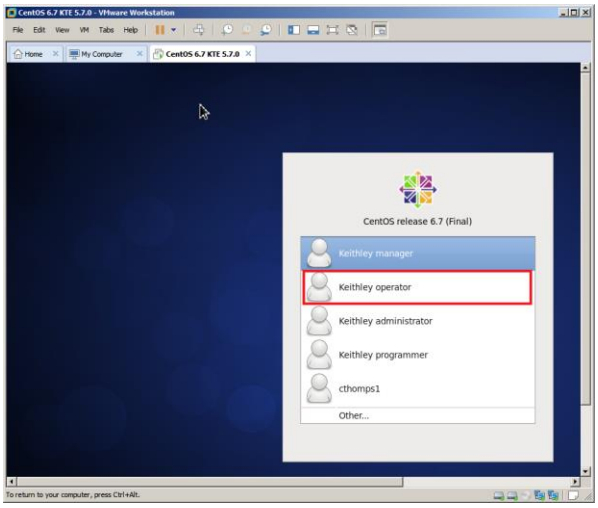
NOTE

You can create operator accounts in the Linux® environment that restrict operator inputs and access to the minimum required to run production. This must be done before using the following procedure to start KRM in operator mode.

To start Keithley Recipe Manager (KRM) in operator mode:

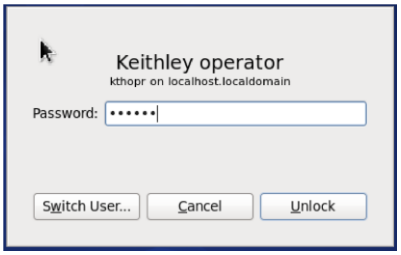
1. On the desktop, select **Keithley operator**.

Figure 106: Logging in as Keithley operator



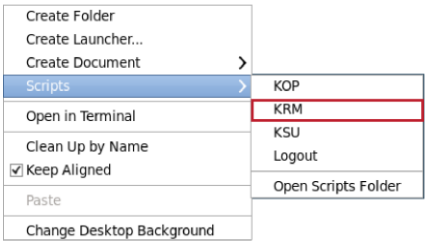
2. At the password prompt, enter your password (if you have not been assigned a password, the default password is `kthopr`).

Figure 107: Keithley operator enter password

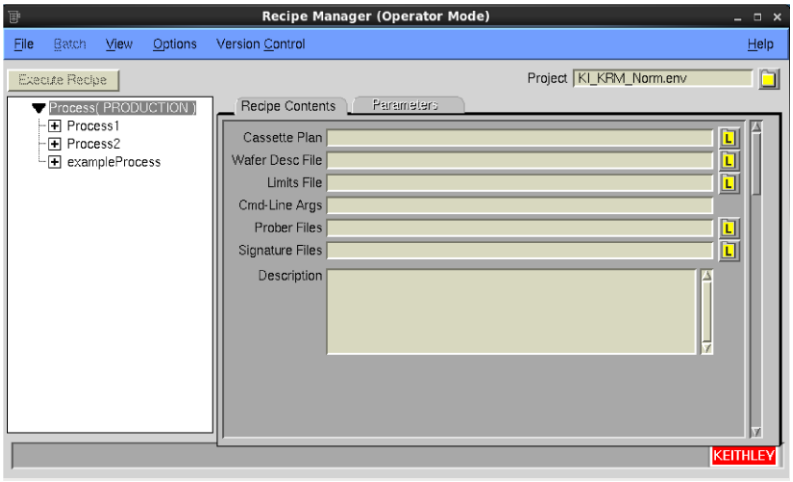


3. Right-click in an open area of the desktop and select **Scripts > KRM** in the shortcut menu that is displayed.

Figure 108: Opening KRM in operator mode



The KRM operator interface opens, as shown in the following figure.



NOTE

After KRM opens, if there is no activity in the user interface for five minutes, the log-in dialog box is displayed. You must enter your password to reopen the interface.

The operator mode has the same menus that the engineering mode has, but some of the menu selections are disabled. Though most user inputs are restricted in this mode, you can still view recipe information on the Recipe Contents and Parameters tabs.

For more information about the KRM interface, see The Keithley Recipe Manager user interface (on page 7-15). For more information about running a recipe, see Selecting and executing recipes in operator mode (on page 7-36).

Starting in engineering mode

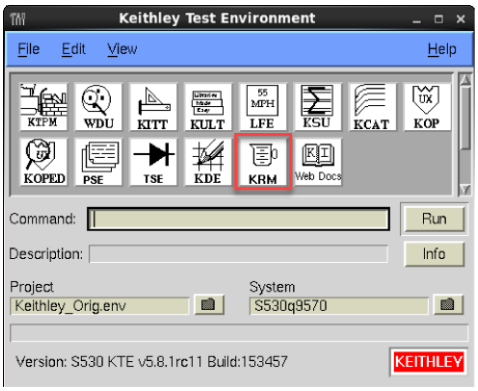
Using the engineering mode gives you full access to all KRM operations.

To start KRM in engineering mode from the command line:

1. Log in to the Linux® environment as Keithley manager.
2. At the command prompt, enter `krm&` and press **Enter**. The Recipe Manager interface opens in engineering mode.

You can also open KRM from the Keithley tools palette. To open the tools palette, type `ktp` at the command prompt.

Figure 109: Open KRM from the Keithley tools palette



Setting up a project

A project is a series of environment variables that point to the directory locations of Keithley Test Environment (KTE) files and user libraries. This collection of directory locations defines a local area (for more information about local areas, see Local areas (on page 8-9)).

NOTE

Projects are used differently in Keithley Recipe Manager (KRM) than in KTE. Projects outside of KRM organize all test-related files, at all revision levels. KRM projects are used only to organize files for test recipe development in a local area. All other files are organized in the archive, staging directories, and tester log using KRM version control.

If you are using the version-control features of KTE and KRM, projects are similar in that user libraries and files must be checked out of the archive before you can make changes. A checked-out file is locked in the archive to prevent concurrent editing of the file.

The KTE installation process automatically generates a single default project. In a multi-user environment, you may want to create multiple projects so different users can develop recipes in personal local areas. You can create additional projects using the `make_project` command-line utility (see Creating a project (on page 7-25) for details).

The following topics describe how to select a project and create and verify a project.

Creating a project

You can create any number of individual projects using the `make_project` command-line utility. You can also create a single personal project when you set up a user account using the `create_user_account` command-line utility (which uses the `make_project` command-line utility).

Use the following format for the `make_project` command-line:

```
make_project <Path> <ProjectName>
```

For example:

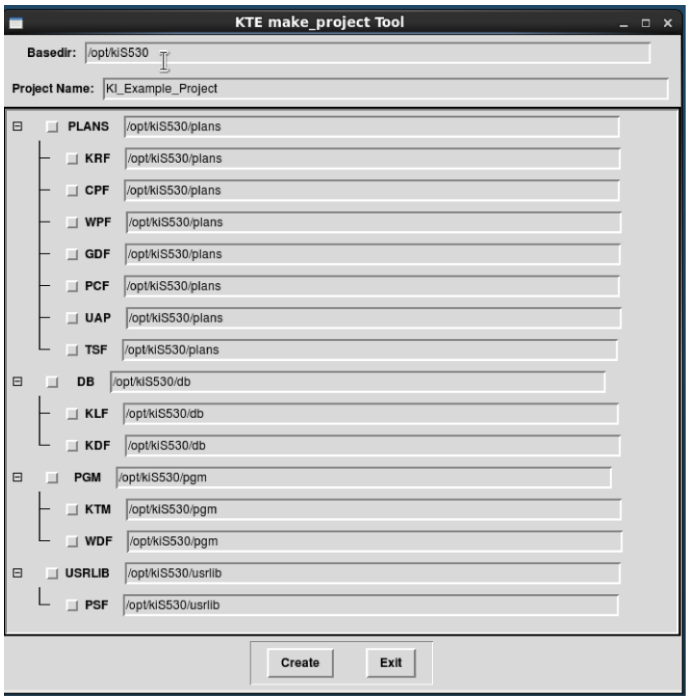
```
make_project $KIHOME KI_Example_Project
```

When you execute the `make_project` script, a screen similar to the following figure is displayed.

NOTE

When the KTE make_project Tool first opens, updated paths are not visible; you must update the paths, as described in the text following this figure.

Figure 110: KTE make_project_tool interface before updating paths

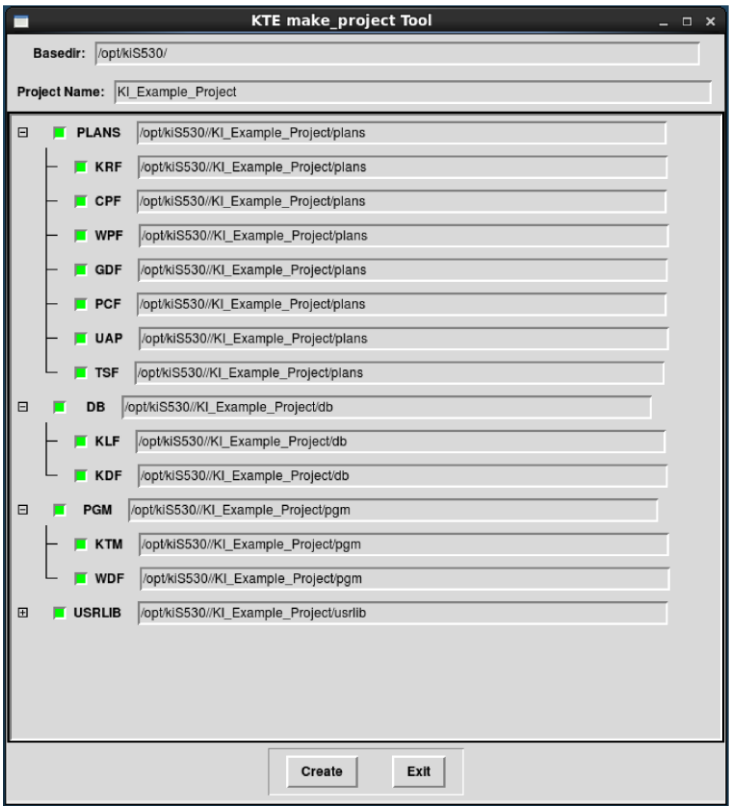


To update the paths, select the square button before each top-level item in the window. When you select a box next to the path in the PLANS, DB, PGM, and USERLIB boxes, the path is automatically populated in the subpaths in the project.

If you want a different path or subpath, select the button next to the path you want to change and edit the text of the existing path as needed.

The following figure shows the interface once you have updated the paths.

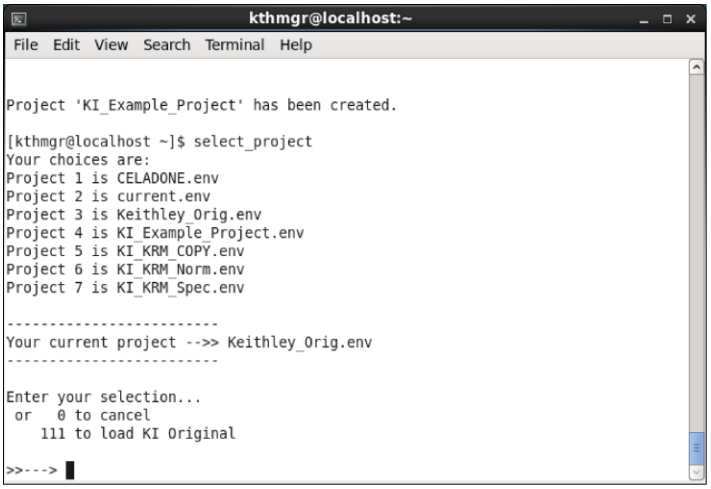
Figure 111: The KTE make_project interface after updating paths



To verify that your project was created:

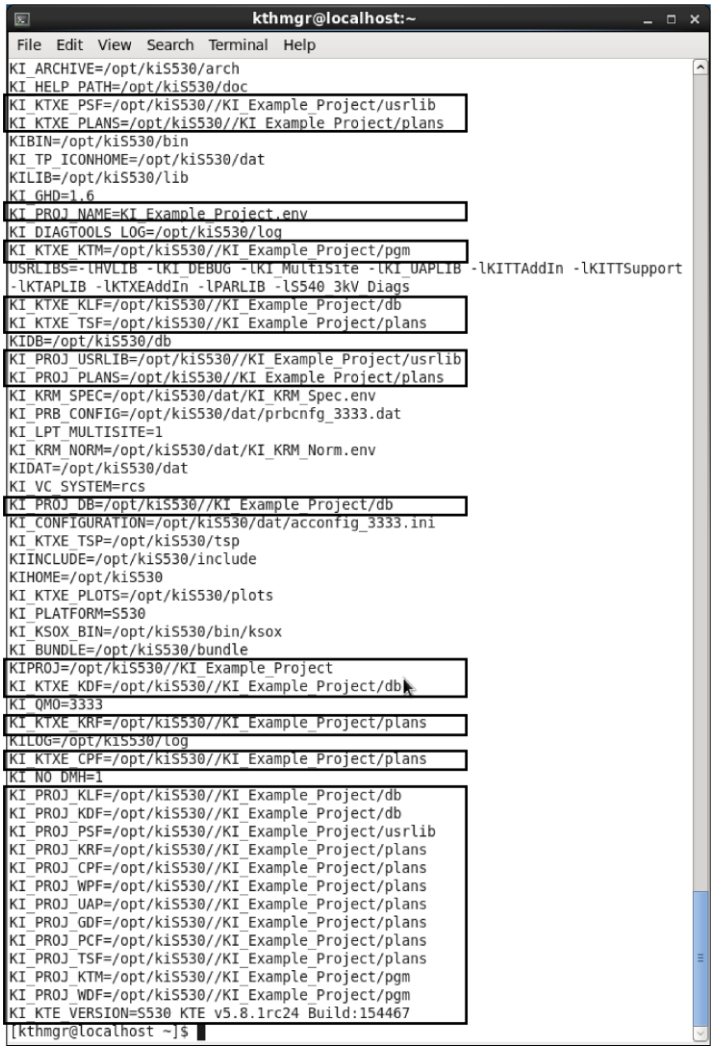
1. At the command prompt, type `select_project`. A list of existing projects is returned; your project should be visible in the list.

Figure 112: Viewing available projects



2. Select the project.
3. Check the environment variables in your project using the `ki?` command. A list of all the project variables and related files is returned, as shown in the following figure.

Figure 113: Project environment variables for the local area



Selecting a project

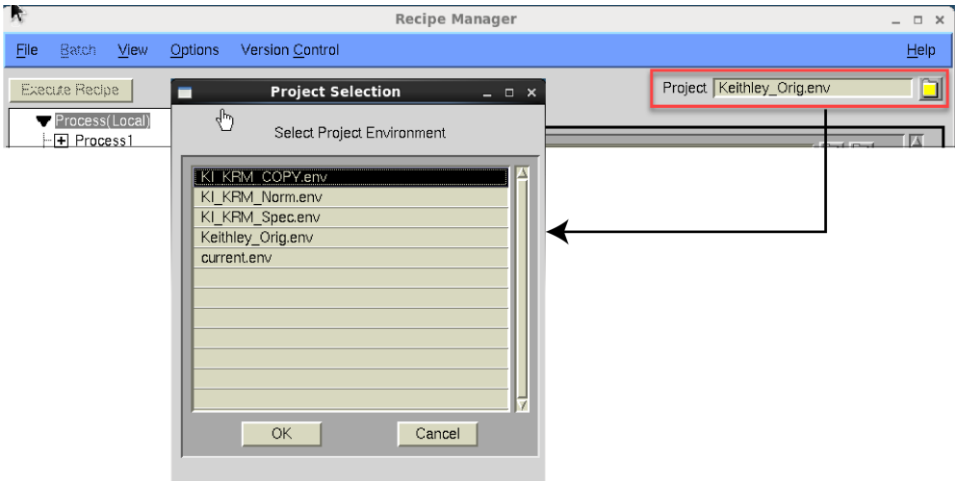
A local area is a directory on a system that is reserved for recipe and supporting file development. File directories in a local area are specified by a project, which lists environment variables that point to the directories.

The Keithley Recipe Manager (KRM) window that opens at startup uses a default project. However, if additional projects have been created, you can select a different one.

To select a project from the KRM interface:

1. In the upper right corner of the KRM interface, select the yellow folder icon next to **Project**. The Project Selection window displays a list of existing projects.

Figure 114: Selecting an existing project

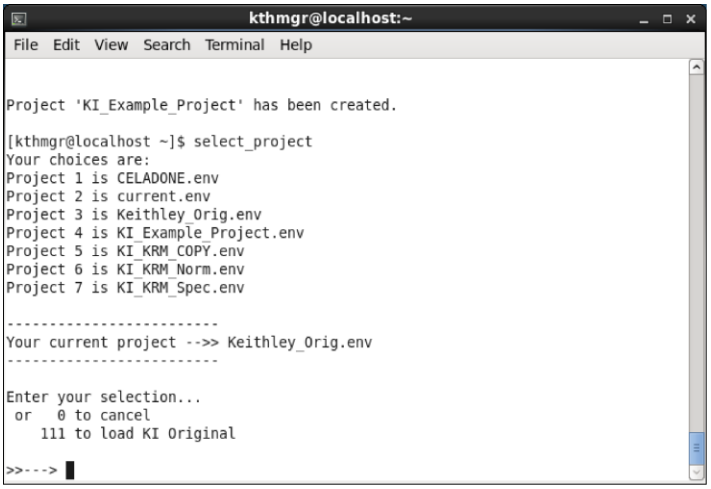


2. Select a project and select **OK**. The name of the selected project is displayed in the Project box and now defines the KRM local area.

To select a project from the command line:

1. At the command prompt, enter `select_project`. A list of available projects is returned.

Figure 115: Viewing available projects



2. Enter the number of the project you want to select at the command prompt.
3. To verify that the project you wanted was selected, open KRM.
4. Look at the project name shown in the upper right side of the interface. The project name you selected should be visible.

Figure 116: Active project

Using Keithley Recipe Manager with KTE tools

When developing a recipe, you may need to create new supporting files or edit existing supporting files that were created in other Keithley Test Environment (KTE) tools. You can access these tools through Keithley Recipe Manager (KRM).

The following KTE tools are accessible directly through Keithley Recipe Manager (KRM):

- Keithley Test Plan Manager (KTPM)
- Wafer Description Utility (WDU)
- Limits File Editor (LFE)
- Keithley Interactive Test Tool (KITT)
- Test Structure File Editor (TSE)
- Keithley User Library Tool (KULT)

You start these tools from KRM by right-clicking the file names on the Recipe Contents tab or Parameters tab. The corresponding file automatically opens when the related tool starts.

For example, right-click a KITT file (`<filename>.ktm`) under KTM Name on the Parameters tab and select **Modify**. The KITT Test Macro Editor opens the `<filename>.ktm` macro.

When you right-click, shortcut menus open that allow you to make the following selections:

- **View:** For all tools except KULT, selecting **View** on the shortcut menu starts the tool and fetches the associated file. A read-only version of the file is opened in the corresponding tool, and the file remains unlocked in the archive (see Fetching a recipe or supporting file into the local area (on page 7-53)).
- **Modify:** For all tools except KULT, selecting the **Modify** option in the shortcut menu starts the tool and checks out the associated file. As a result, the file is read-write in the tool and is locked in the archive (See also Checking out a recipe or supporting file to the local area for revision (on page 7-51)).
- **Start KULT:** For KULT only, right-clicking a file name under Module Name or Library Name and selecting **Start KULT** opens the selected file in KULT.

Accessing KTPM, WDU, and LFE from the Recipe Contents tab

You can open the Keithley Test Plan Manager (KTPM), Wafer Description Utility (WDU), and Limits File Editor (LFE) tools from the Keithley Recipe Manager (KRM) Recipe Contents tab.

To open KTPM, WDU, and LFE from the KRM Recipe Contents tab:

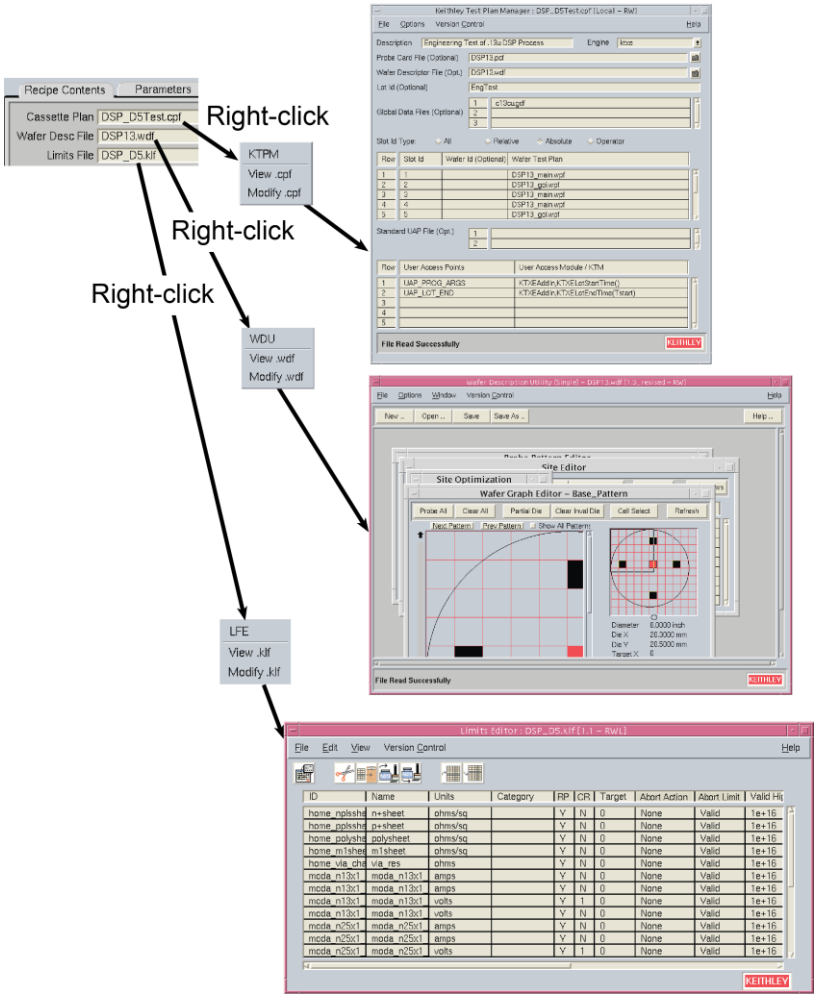
1. Right-click on the file name of the file that you want to open. A shortcut menu for the related KTE tool is displayed. See the following figure for an example.

Figure 117: KTPM, WDU, and LFE shortcut menus

2. In the shortcut menu, select one of the following:
   - Select **View** to open (fetch) a read-only copy of the file in the local area.
   - Select **Modify** to check out the file and lock it in the archive.

   The related tool opens as specified.

Figure 118: Accessing KTPM, WDU, and LFE from the Recipe Contents tab



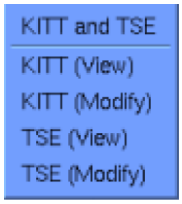
Accessing KITT and TSE from the Parameters tab

You can access the Keithley Interactive Test Tool (KITT) and Test Script Editor (TSE) tools from the Keithley Recipe Manager (KRM) Parameters tab.

To access KITT and TSE from the Parameters tab:

1. On the Parameters tab, right-click the name of the `.ktm` file you want to open. The following shortcut menu is displayed.

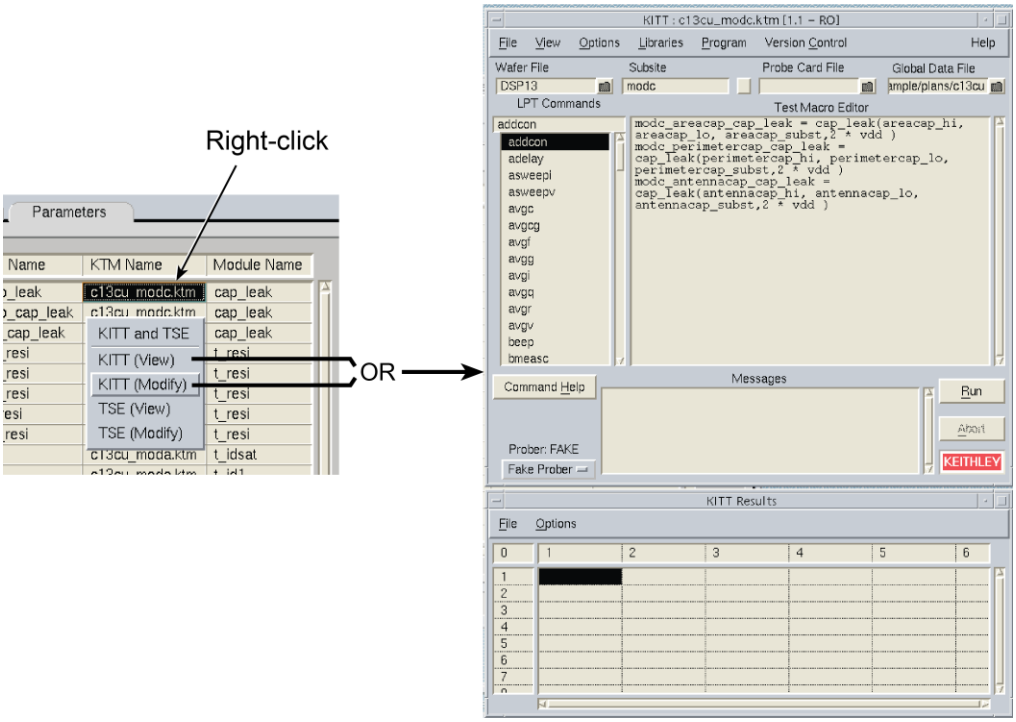
Figure 119: Shortcut menu for KITT and TSE access



2. In the shortcut menu, select one of the following:
   - Select **View** to open (fetch) a read-only copy of the file in the local area.
   - Select **Modify** to check out the file and lock it in the archive.

   The related tool opens as specified.

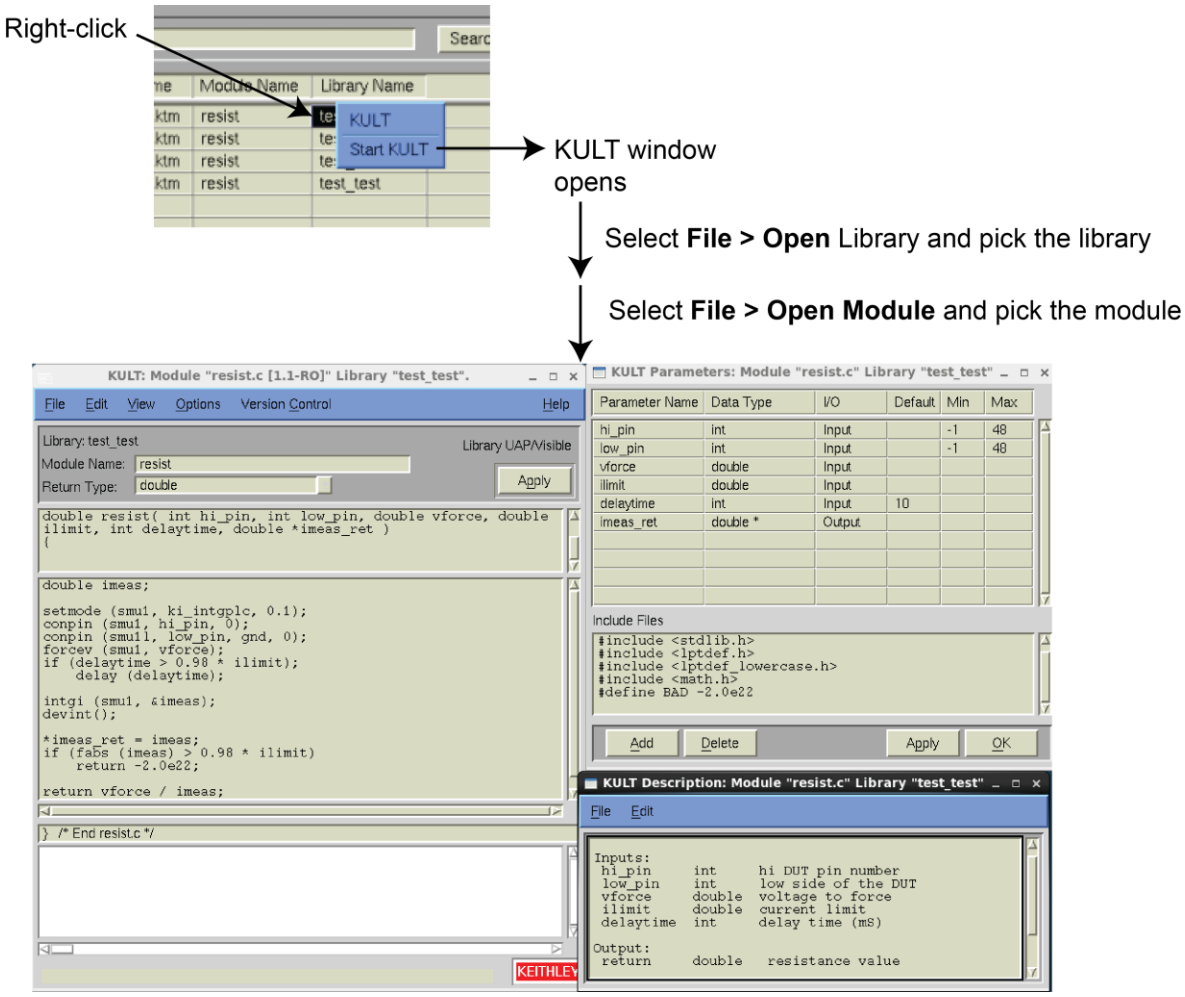
Figure 120: Accessing KITT from the Parameters tab



Accessing KULT from the KRM Parameters tab

You can open the Keithley User Library Tool (KULT) from the Keithley Recipe Manager (KRM) Parameters tab by right-clicking a module name or library name and selecting **Start KULT**. The module or library that you right-clicked is opened in KULT. Once KULT is open, you can use the File menu options to create, open, copy, or delete a library or module.

Figure 121: Accessing KULT from the KRM Parameters tab



Searching for files from the KRM Parameters tab

You can use the Parameters tab in Keithley Recipe Manager (KRM) to search for a parameter name in the list of parameters on the tab. This is useful when you have a long list of parameters in your recipe. You can then launch the Keithley Interactive Test Tool (KITT) to view or edit the parameter.

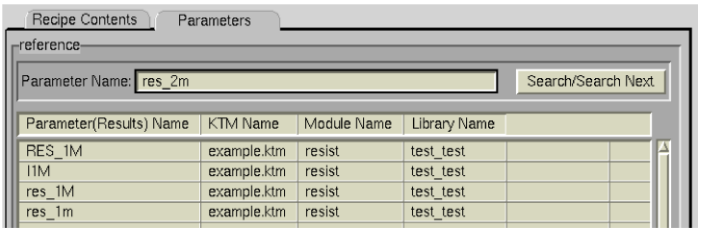
To search for a parameter:

1. In the View menu, select **Parameter List**. The contents of the Parameters tab is displayed.
2. In the box next to Parameter Name, type part of the parameter name followed by an asterisk (*) and select **Search/Search Next**. KRM searches the Parameter(Results) Name column for matching values and highlights the first matching parameter.

NOTE

The Parameter Name search box accepts all alphanumeric characters and the underscore (_) character. You can use the ? and * characters as wild-card values.

Figure 122: Searching for parameters on the KRM Parameters tab



3. To search for another instance of the same parameter, select **Search/Search Next** again.

NOTE

The Parameter Name value you entered remains in the search box when you select a different recipe name. This allows you to search for parameters in several recipes without reentering the search value.

Selecting and executing recipes in operator mode

You can use Keithley Recipe Manager (KRM) instead of the Keithley Operator (KOP) Utility as the operator interface for launching test plans. The system that an operator will use can be set up so that KRM is the default program. The following topics describe the operator mode and how to select and execute recipes in this mode.

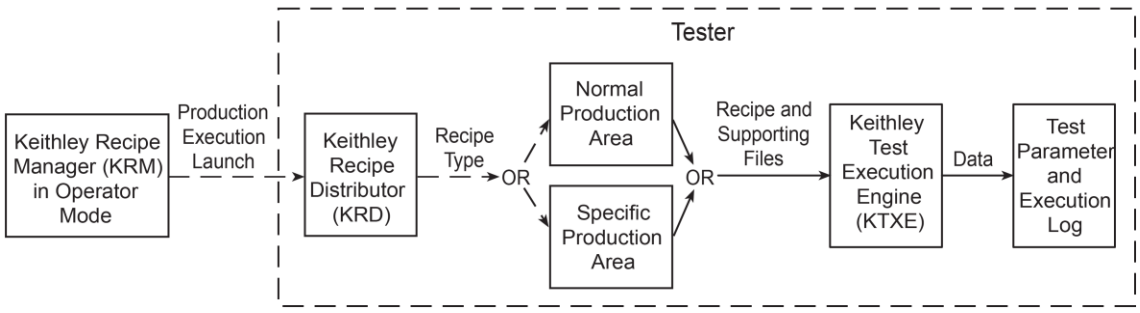
NOTE

Using conventional Linux® capabilities, operator accounts can be set up that restrict operator inputs and access rights to the minimum required to run production (generally, operator mode restricts available functions to the selection and execution of recipes). The following topics assume that the necessary accounts have been set up, that KRM is launched from the Keithley Tool Palette or as the default program, and that KRM starts specifically in operator mode.

The recipe execution process in operator mode

The following figure illustrates execution of a recipe in Keithley Recipe Manager (KRM) operator mode. A description of the process follows the figure.

Figure 123: Recipe execution using KRM in Operator mode



Recipe execution in operator mode proceeds as follows:

1. Log in and start KRM in operator mode.
2. In KRM, select a recipe using the recipe navigator, which displays a graphical hierarchy of processes, product families, and recipes.
3. Once the recipe is selected, select **Execute Recipe**, which initiates the following:
   - Loads the selected recipe from the normal or specific production directory.
   - Runs the Keithley Test Execution Engine (KTXE), generating output files.

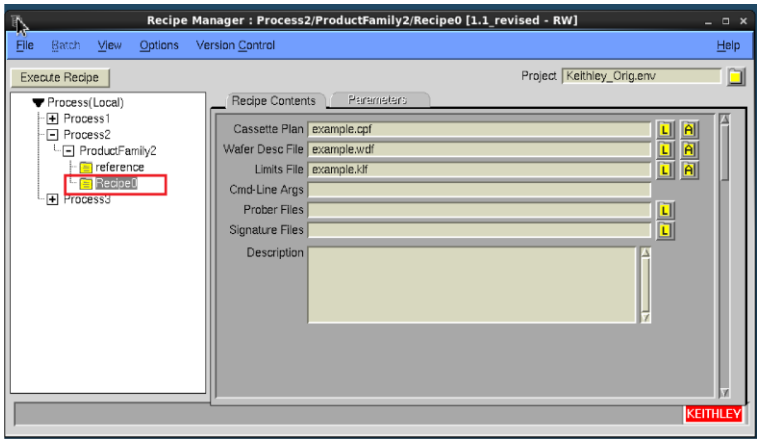
The above sequence can also be executed using the automation link option. However, normal and specific recipes must be created in KRM, which cannot be done over the automation link.

Executing a recipe in operator mode

To execute a recipe in operator mode:

1. In the recipe navigator, expand the process and product family that are related to the recipe you need to run.
2. Select the recipe to be executed (a recipe must be selected before the Execute Recipe button is enabled).

Figure 124: Select a recipe to execute

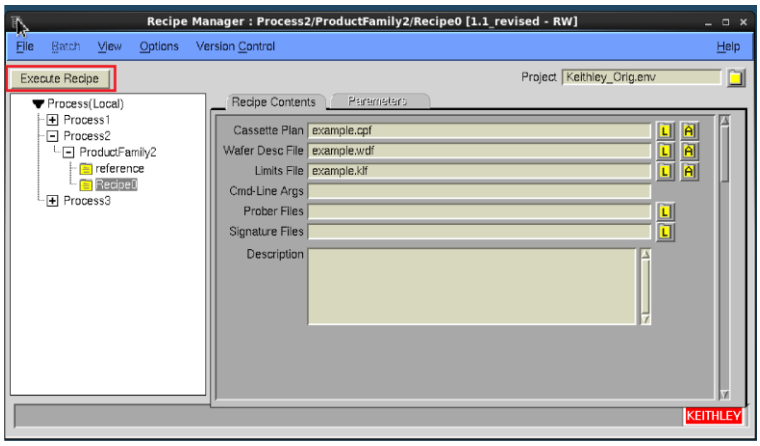


NOTE

To see recipes in the recipe navigator that have been added after starting KRM, select **Refresh Recipe List** in the View menu.

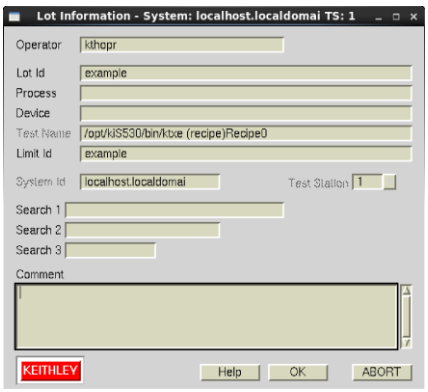
3. Select **Execute Recipe**.

Figure 125: Execute the recipe



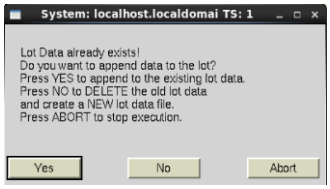
4. In the Lot Information dialog box that opens, enter any additional information or comments for the execution of this recipe and select **OK**.

Figure 126: Lot Information dialog box



5. If lot data already exists, a dialog box is displayed that asks you if you want to append data to the lot. Select **Yes** to append to the existing lot data or **No** to delete the old lot data and create a new lot data file.

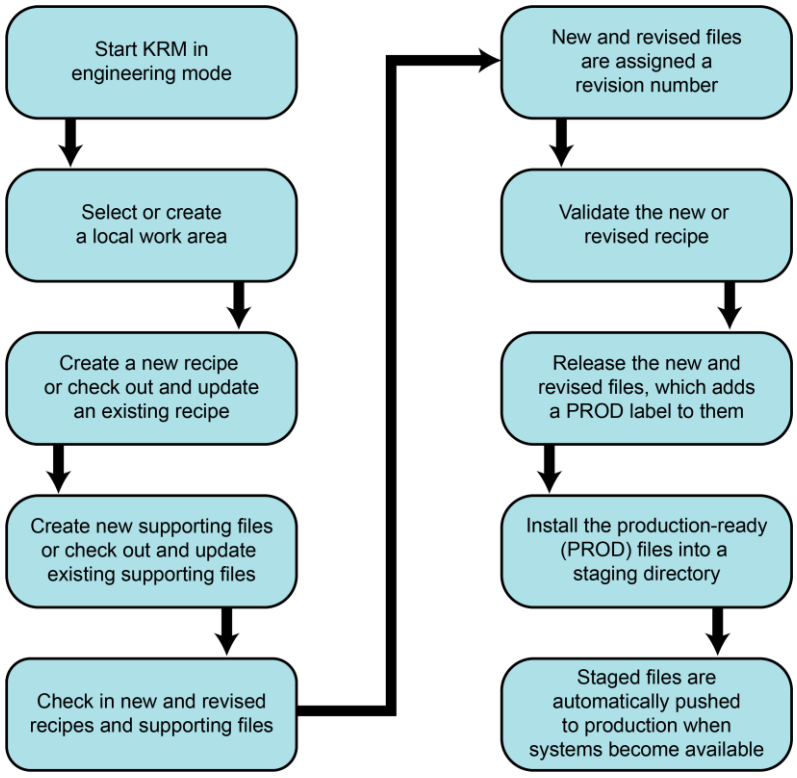
Figure 127: Append data to existing lot dialog box



The Keithley Recipe Manager engineering mode process

The Keithley Recipe Manager (KRM) engineering mode workflow typically follows the process in the following figure.

Figure 128: Typical recipe development process



The following topics describe this process in detail.

To get familiar with how the individual tasks work as complete process, see the Examples (on page 7-70) later in this section:

- Example 1: Create a new recipe using existing files (on page 7-71)
- Example 2: New recipe and wafer description file from existing files (on page 7-77)
- Example 3: Release and install a recipe (on page 7-86).

Each of these examples builds on the example before it, giving you a way to try out the entire process.

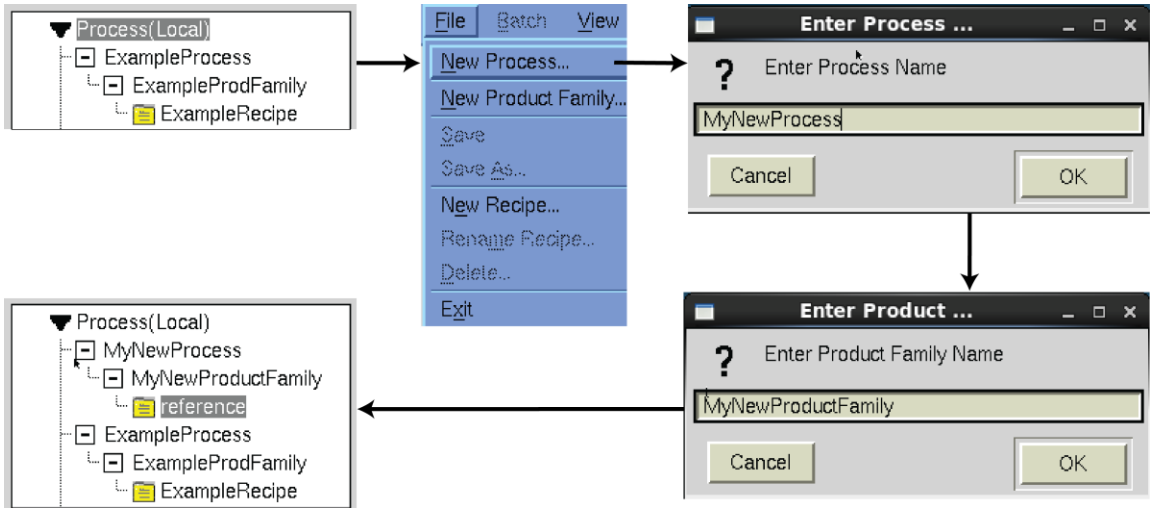
Creating a new process and product family

Recipes in Keithley Recipe Manager (KRM) are organized in the local area under a process (a group of products sharing common testing requirements) and product family (specific model). Before creating a new recipe, you must create a new process and process family.

To create a new process and product family:

1. In the recipe navigator, select **Process(Local)**.
2. Select **File > New Process.**
3. In the dialog box that opens, enter the name of the new process.
4. Another dialog box opens; enter the name of a new product family.
5. Select **OK**. A new process, a new product family, and an empty reference recipe are displayed in the recipe navigator.

Figure 129: Creating a new process



NOTE

To populate the new reference recipe, select a cassette plan file (`.cpf`), wafer description file (`.wdf`), and limits file (`.klf`) using the local (L) and archive (A) icons (see Revising a recipe (.krf file) (on page 7-55)). Enter information in other Recipe Contents fields if appropriate. You can also create new supporting files; see Creating new supporting files (on page 7-47) for details.

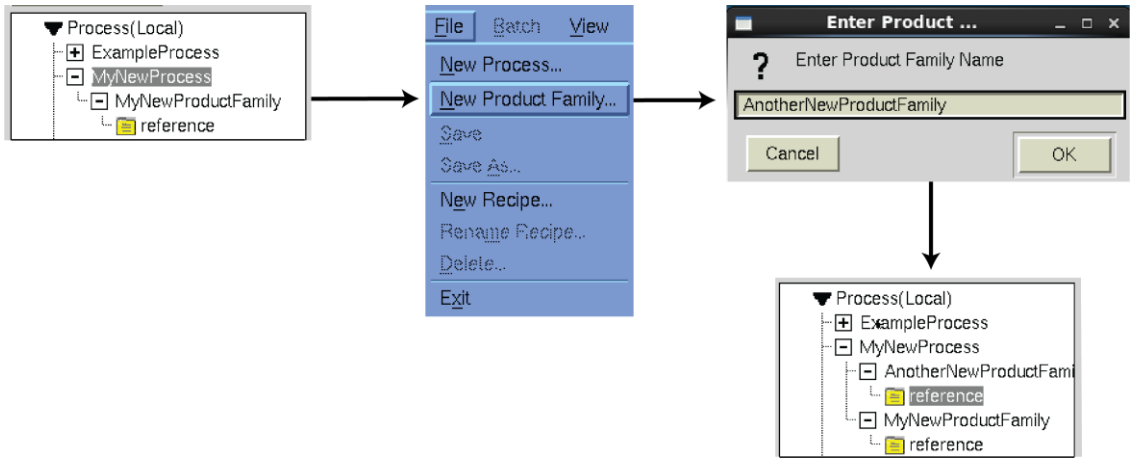
Creating a new product family under an existing process

You can have multiple product families (specific product models) under a process (a group of products sharing common testing requirements).

To create a new product family under an existing process:

1. In the recipe navigator, select the process that the new product family is associated with.
2. Select **File > New Product Family**.
3. In the dialog box that opens, enter the name of the new product family.
4. Select **OK**. The new product family and an empty reference recipe are displayed in the recipe navigator.

Figure 130: Creating a new product family under an existing process



NOTE

To populate the new reference recipe, select a cassette plan file (`.cpf`), wafer description file (`.wdf`), and limits file (`.klf`) using the local (L) and archive (A) icons (see Revising a recipe (.krf file) (on page 7-55)). Enter information in other Recipe Contents fields if appropriate.

Creating a new recipe

Recipes are organized under a process (products with similar test requirements) and product family (specific model) in Keithley Recipe Manager (KRM). You can create a new recipe under an existing process-product family combination, or you can create a new process-product family combination for your recipe.

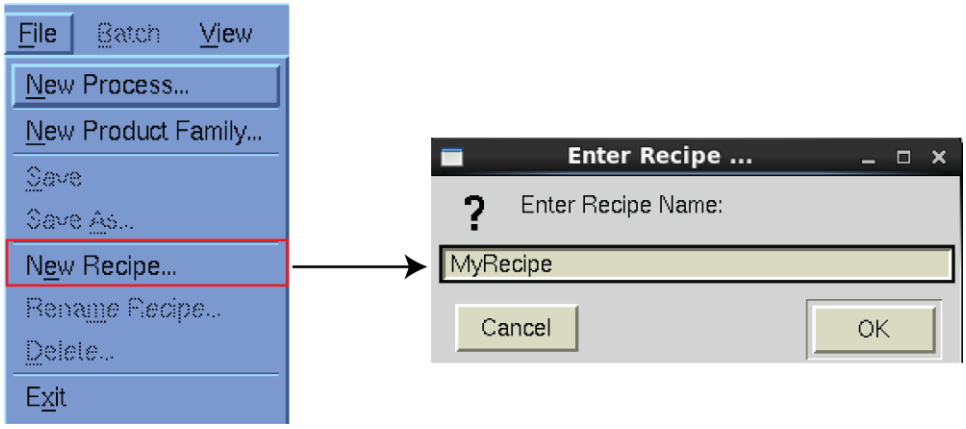
If you are starting with no existing supporting files (for example cassette plan files (.cpf), wafer description files (`.wdf`), limits files (`.klf`), and so on), you can create completely new files using Keithley Test Environment (KTE) tools. For more information about creating supporting files in KTE, see Creating new supporting files (on page 7-47).

You can also create a new recipe using existing supporting files. The following instruction assumes you have already created the new supporting files or you are planning to use existing supporting files.

To create a new recipe:

1. In the KRM interface, select the new product family you created or an existing product family.
2. Select **File > New Recipe**. The Enter Recipe dialog box opens.

Figure 131: Create a new recipe



3. Enter the new recipe name.

NOTE

A complete recipe name includes the process, product family, and recipe name (for example, `Process-A__ProdFamily-A1__Recipe-A1.krf)`. This combination must be unique. For more information about naming recipes, see Naming recipes (on page 7-43).

4. Select **OK**. The new recipe is displayed in the recipe navigator. The recipe is populated as follows:
   - If the product family that was selected in step 1 contains a reference recipe, the new recipe is a copy of the reference recipe.
   - If the product family that was selected in step 1 does not contain a reference recipe, the new recipe is empty.

NOTE

The new recipe is read-write and is not locked because no previous version of it exists in the archive.

5. Select the cassette plan, wafer plan, and limits supporting files on the Recipe Contents tab. You can select files from the local area (yellow folder labeled with an L) or the archive (yellow folder labeled with an A).
6. Enter other information in other Recipe Contents and Parameters tab fields if appropriate.
7. When you are done populating the recipe, select **File > Save**.
8. If the recipe is ready to be assigned a revision number, check in the recipe (you must check in the new recipe to place it in the archive). For more information, see Checking files into the archive (on page 7-58).

Naming recipes

When you name a recipe, Keithley Recipe Manager (KRM) prepends the process and product family names to the name you specify, which creates the complete recipe name (for example, `Process-A__ProdFamily-A1__Recipe-A1.krf`). This combination of names must be unique. However, you can repeat the product family-recipe name combination (`ProdFamily-A1__Recipe-A1.krf`) under any number of separate processes.

If multiple recipes exist with the same process-product family-recipe name combination, you will receive an error message and a request to provide a unique recipe identity.

If you want to require that every recipe name and product family combination be unique, you can edit the `$KIHOME/krm.ini file`. Open the file, and next to the `forceUniqueRecipeNames` entry in the `<COMMON>` section of the file, change the value from `n` to `y`.

Creating a new recipe under an existing product family

To create a new recipe under an existing product family:

1. Select a product family.
2. Select **File > New Recipe**. An Enter Recipe dialog box is displayed.
3. Enter the new recipe name.

NOTE

A complete recipe name includes the process, product family, and recipe name (for example, `Process-A__ProdFamily-A1__Recipe-A1.krf)`. This combination must be unique. For more information about naming recipes, see Naming recipes (on page 7-43).

4. Select **OK**. The new recipe is displayed in the recipe navigator under the product family you selected. The recipe is populated as follows:
   - If the product family that was selected in step 1 contains a reference recipe, the new recipe is a copy of the reference recipe.
   - If the product family that was selected in step 1 does not contain a reference recipe, the new recipe is empty.

NOTE

The new recipe is read-write and is not locked because no previous version of it exists in the archive.

5. Select the cassette plan, wafer plan, and limits supporting files on the Recipe Contents tab. You can select files from the local area (yellow folder labeled with an L) or the archive (yellow folder labeled with an A).
6. Enter other information in other Recipe Contents and Parameters tab fields if appropriate.
7. When you are done populating the recipe, select **File > Save**.
8. If the recipe is ready to be assigned a revision number, check in the recipe (you must check in the new recipe to place it in the archive). For more information, see Checking files into the archive (on page 7-58).

Creating a new recipe using an existing recipe

To create a new recipe using an existing recipe:

1. Select an existing recipe (you do not need to check out the recipe).
2. Select **File > Save As**. The Enter Recipe Name dialog box is displayed.
3. Enter the new recipe name and select **OK**. The new recipe is added to the recipe navigator.

NOTE

A complete recipe name includes the process, product family, and recipe name (for example, `Process-A__ProdFamily-A1__Recipe-A1.krf)`. This combination must be unique. For more information about naming recipes, see Naming recipes (on page 7-43).

4. Select one of the yellow folder icons next to Cassette Plan, Wafer Desc. File, and Limits File as needed to select a different version of the file from the local area (folder labeled L) or the archive (folder labeled A).
5. Revise information in other recipe contents fields if appropriate. See Revising a recipe (on page 7-55) for additional information.
6. When you have completed the modifications, select **File > Save** to save the new recipe.
7. When the recipe is ready to be assigned a revision number, select **Version Control > Other Operations > Check In**. The new recipe is placed in the archive. For more information about checking files into the archive, see Checking files into the archive (on page 7-58).

For an example of this process, see Example 1: Create a new recipe using existing files (on page 7-71).

Creating a specific recipe

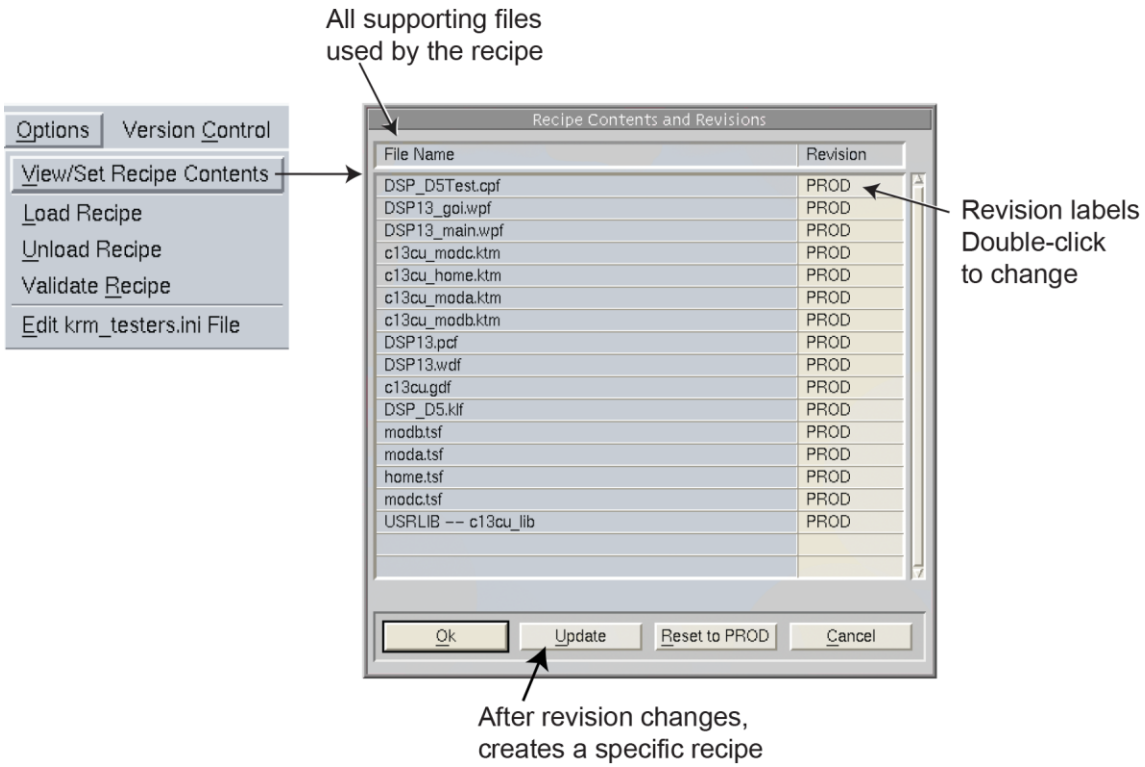
In Keithley Recipe Manager (KRM), there two types of recipes:

- **Normal recipe:** A normal recipe only calls files labeled for production (PROD).
- **Specific recipe:** A specific recipe calls at least one non-PROD labeled version of a supporting file, commonly for diagnostic purposes. Specific recipes are usually created from an existing recipe, with a different version of some supporting files.

To create a specific recipe:

1. Select an existing normal (PROD-labeled) recipe.
2. Select **Options > View/Set Recipe Contents**. The Recipe Contents and Revisions dialog box is displayed.

Figure 132: The Recipe Contents and Revisions dialog box



3. Double-click the revisions of one or more PROD-labeled supporting files to change them to non-PROD labeled versions.
4. Select **Update** to create the specific recipe. A copy of the recipe and its supporting files is saved in the local area.
5. When prompted, select **Yes** to load the recipe and the specified versions of the supporting files into your local area. This is necessary so that you can later validate the recipe.

CAUTION

If a supporting file is being worked on in the local area when you load the recipe, it is overwritten by the version of the supporting file you selected for the specific recipe.

NOTE

The process of creating the specific recipe loads one or more non-PROD labeled supporting files into the local work area. These supporting files are used by the normal recipe that was copied to make the specific recipe. They also may be used by other recipes. Before working on another recipe, be sure to select **Options > Load Recipe** to make sure that the correct versions of its supporting files are present in the local area.

Creating new supporting files

There are several ways to create new supporting files and user libraries for recipes:

- Create the new supporting files using the Keithley Test Environment (KTE) tools and then select them in Keithley Recipe Manager (KRM). See the Software (on page 6-1) section of this manual for detailed information about using KTE tools.
- You can open KTE tools through the KRM interface to create new supporting files or modify and save existing files with new names.

See Revising supporting files and user libraries (on page 7-57) for more information about using existing files to create new supporting files.

To create completely new supporting files using KTE tools through KRM:

1. In KRM, select the recipe that you want to create a new supporting file for.
2. On the Recipe Contents tab, right-click in the empty box next to the file type that you want to create. A shortcut menu is displayed.
3. Select the **View** option for the related KTE tool. The tool opens to an unpopulated screen.
4. Populate the file with appropriate information, as described in the Software (on page 6-1) section of this manual.
5. Select **File > Save** to save a completely new supporting file or **File > Save As** to save a modified supporting file with a new name in the appropriate location.
6. When the supporting file is ready to be assigned a revision number, select **Version Control > Other Operations > Check In**. The new supporting file is placed in the archive and is available for use in multiple recipes. For more information about checking files into the archive, see Checking files into the archive (on page 7-58).

Copying from read-only modules

You can copy the content of read-only modules in the Keithley Interactive Test Tool (KITT) and the Keithley User Library Tool (KULT) by clicking the middle button of the mouse.

Validating a recipe

NOTE

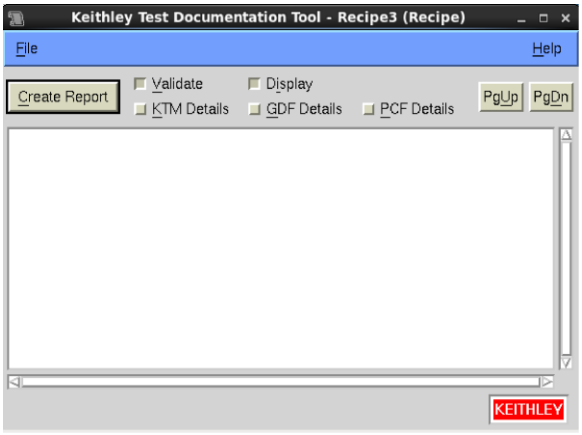
Before validating a recipe, all of its supporting files must be loaded to the local area. See Loading all needed supporting files into the local area at once (on page 7-52).

A new or changed recipe should be validated before being released for production.

To validate a recipe:

1. In Keithley Recipe Manager (KRM), select **Options > Validate Recipe**. The Keithley Test Documentation Tool (KTDT) opens.

Figure 133: KTDT recipe validation and reporting window



2. In the Keithley Test Documentation Tool window, you can select several options:
   - **Validate:** Select this to validate the recipe.
   - **Display:** Select this to display the results of the validation test in the Keithley Test Documentation Tool window.
   - **KTM Details, GDF Details, and PCF Details:** Select any of these options to display information about the recipe and related files in the Keithley Test Documentation Tool window.
3. Select **Create Report.** The recipe is validated, and the selected details are visible in the Keithley Test Documentation Tool window.

CAUTION

When prompted to load recipe files before validating, do not select Yes unless you are sure that the checked-out supporting files in the local area may be safely overwritten.

Using the Keithley Test Documentation Tool, the software checks the usability of a recipe without hardware execution. It verifies:

- Syntax in recipe elements
- Presence of referenced files (and their versions) in specified locations
- Availability of user access point (UAP) code
- The presence and validity of parameters

For more information about the Keithley Test Documentation Tool, see the Software (on page 6-1) section of this manual.

NOTE

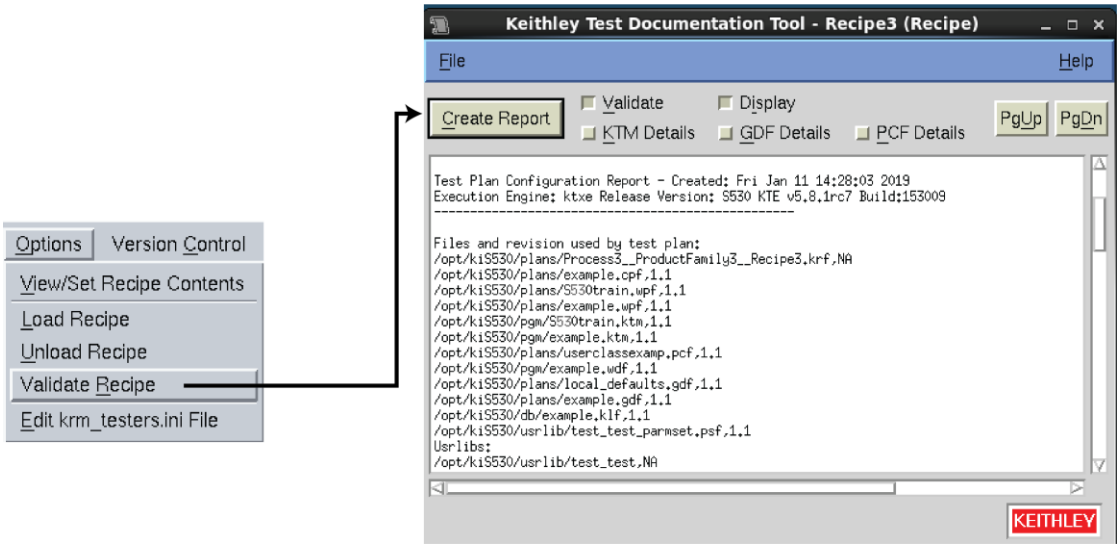
The steps initiated by the combined-action **Version Control > Deliver** and **Version Control > Other Operations > Release** menu selections (in the KRM window only) include the **Validate Recipe** action as an option. Refer to Releasing files for production for more information.

KRM performs the **Validate Recipe** process automatically when a production run is requested. If KRM finds that the recipe is invalid, it notifies the operator and prevents execution.

Recipe validation reports

The recipe validation report contains information about the contents of the recipe and version numbers of files.

Figure 134: Recipe validation report



See Version control in KRM (on page 7-92) for more information about version control in Keithley Recipe Manager. For detailed information about version control, see the Version control (on page 8-1) section of this manual.

Editing existing recipes

The following topics describe the tasks necessary to revise a recipe.

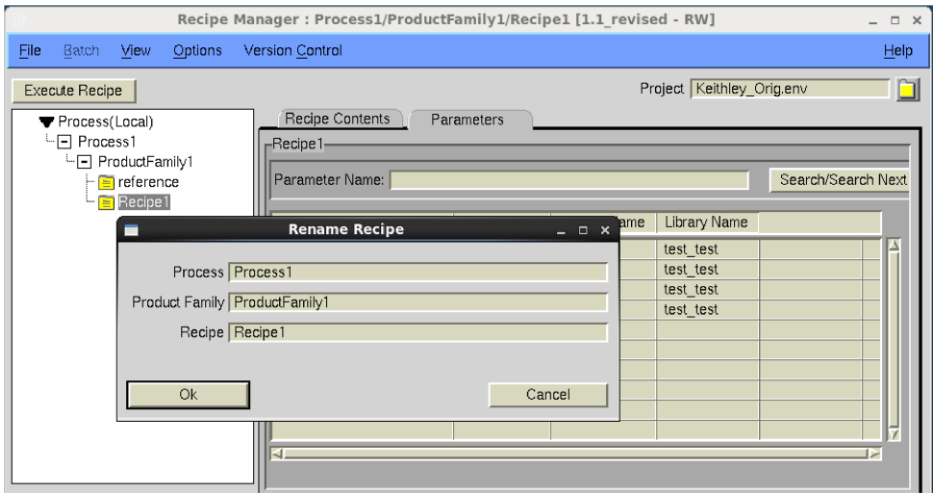
Renaming a recipe

You can rename a recipe in Keithley Recipe Manager (KRM). When you rename a recipe, you can also rename the process and product family.

To rename a recipe:

1. Select the recipe you want to rename.
2. If the recipe is not checked out of the archive, select **Version Control > Other Operations > Check Out**.
3. Select **File > Rename Recipe**. The Rename Recipe dialog box opens, allowing you to change all three components of the recipe name as needed.

Figure 135: Rename Recipe dialog box



NOTE

A complete recipe name includes the process, product family, and recipe name (for example, `Process-A__ProdFamily-A1__Recipe-A1.krf`). This combination must be unique. For more information about naming recipes, see Naming recipes (on page 7-43).

4. Rename the recipe with a unique combination of recipe name, process, and product family and select **OK**. KRM validates the new name and prompts you to verify that you want the recipe to be adjusted and updated in the archive.

5. Select **Ok** to accept the changes.
6. If a PROD label exists for the recipe, enter the release password to authorize sending the updates to the production environment. The default password is `kthadm`.

KRM creates a copy of the originally named recipe with the new name in the local area. If the originally named recipe file is in the archive, KRM checks in both the original file and the renamed copy of the file and generates a `Renamed from [old name] to [new name]` log message.

If the original recipe name had a PROD label, KRM removes the PROD label from the original recipe and applies it to the renamed copy of the recipe.

For multiple networked systems, KRM places a `DELETE--oldName.cmd` file in the staging directory for each client. This causes the originally named recipe to be deleted from the client during the next file installation cycle.

KRM then deletes the originally named recipe from the local area and refreshes the recipe navigator to show the new name.

Checking out recipes and supporting files to the local area for revision

Before you can modify an existing recipe or supporting file, you must check it out of the archive.

To check a recipe out of the archive:

NOTE

The Check Out operation applies to only one file at a time. You cannot check out multiple files with a single Check Out operation.

1. Select **Version Control > Other Operations > Check Out**. The Check Out Revision dialog box is displayed.
2. Select the version that you want to open:
   - **Head:** The most recent version of a file; usually the version used for production (PROD).
   - **PROD:** The version of the file that is released for production; the head and PROD files are often the same version.
   - **Other:** A version of the file that is not a head or PROD version; can be used for troubleshooting changes between versions of files.
3. Select **Ok**. KRM loads a copy of the file in the local area and places a lock on the archive copy so that no one else can concurrently modify that file version.

You can keep a file locked in the archive over multiple work sessions. A checked-out file remains locked until it is checked in. Exiting Keithley Recipe Manager (KRM) does not change the status.

NOTE

The Version Control > Modify combined-action menu item (an option in Keithley Recipe Manager and all Keithley Test Environment (KTE) tools except the Keithley User Library Tool (KULT)) implicitly includes the Check Out action. However, Modify automatically checks out the PROD-labeled version. You are not given a choice of PROD, Head, or Other.

For more information about the Check Out and Modify menu items, see Version Control menu (on page 7-18).

Loading all needed supporting files into the local area at once

Before you can verify and install a selected recipe in the recipe navigator, all supporting files for the recipe must be in the local area. You can load a PROD-labeled copy of all necessary files to the local area at once using the Load Recipe option.

NOTE

You do not need to check out the files before selecting Load Recipe. However, before modifying any of these files, you must check out the individual file you want to change. A loaded file is read-only until you check it out.

To load a recipe and its supporting files in a single operation:

1. In Keithley Recipe Manager (KRM), select **Options > Load Recipe**. If there already is a writable version of one or more of the files in the local area, you are prompted to choose whether to overwrite them.
2. Select **Yes** to continue the process. When the load is complete, `Recipe successfully loaded` is displayed.
3. Select OK to close the dialog box.

NOTE

Selecting **Version Control > Modify** implicitly includes the **Load** and **Check Out** actions for the selected recipe. This is available in KRM and all Keithley Test Environment tools except the Keithley User Library Too (KULT), where it is available in some submenus.

Fetching a recipe or supporting file into the local area

A fetch operation copies a single recipe or supporting file into the local area from the archive, without locking it. A fetched recipe or supporting file is read-only.

To fetch a recipe:

1. In the Keithley Recipe Manager, select the recipe that you want to fetch.
2. Select **Version Control > Other Operations > Fetch**. The Fetch Revision dialog box is displayed
3. Select the version of the recipe you want to fetch:
   - **Head:** The most recent version of a file; usually the version used for production (PROD).
   - **PROD:** The version of the file that is released for production; the head and PROD files are often the same version.
   - **Other:** A version of the file that is not a head or PROD version; can be used for troubleshooting changes between versions of files.

   A read-only copy of the file is inserted into the local area. The file remains unlocked in the archive.

NOTE

In archive view, selecting a recipe automatically results in a fetch operation. If the recipe was not previously present in your local area, it is now present when you switch back to the local view, being displayed in the recipe navigator. See also Adding a new recipe to the local area from the archive (on page 7-54).

To fetch a supporting file into the local area:

1. In Keithley Recipe Manager, select the recipe in which the supporting file is used.
2. Right-click the supporting file name in the Recipe Contents or Parameters tab to open a shortcut menu.
3. Select the appropriate tool (see Using Keithley Recipe Manager with KTE tools (on page 7-31) for more information). The selected tool opens.

NOTE

Not all supporting files and tools can be opened directly from KRM. Some secondary tools and supporting files must be opened indirectly using menus, buttons, and selections in the primary tools that you can open through KRM.

4. When you right-click a file to start a tool, the file automatically opens when the tool starts.[6] For example, when you start the Keithley Interactive Test Tool (KITT) by right-clicking `c13cu_modc.ktm` on the Parameters tab, the `c13cu_modc.ktm` macro opens in the KITT Test Macro Editor window.
5. Select **Version Control > Other Operations > Fetch**. A read-only copy of the supporting file is placed in the local area, without locking the file in the archive.

Adding a new recipe to the local area from the archive

When a new recipe is checked in to the archive, you can add the recipe to the recipe navigator (local area) without locking the version in the archive.

To add a new recipe to the local area from the archive:

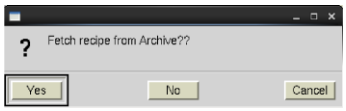
1. In Keithley Recipe Manager (KRM), select **View > Archive View** to switch to archive view.

NOTE

When you have Archive View selected, the recipes that are visible in the recipe navigator are the head (latest) version of the recipes. You cannot check out or modify a recipe or supporting file directly from the Archive View.

2. In the recipe navigator, select the recipe that you want to add to the local area. A fetch dialog box is displayed.

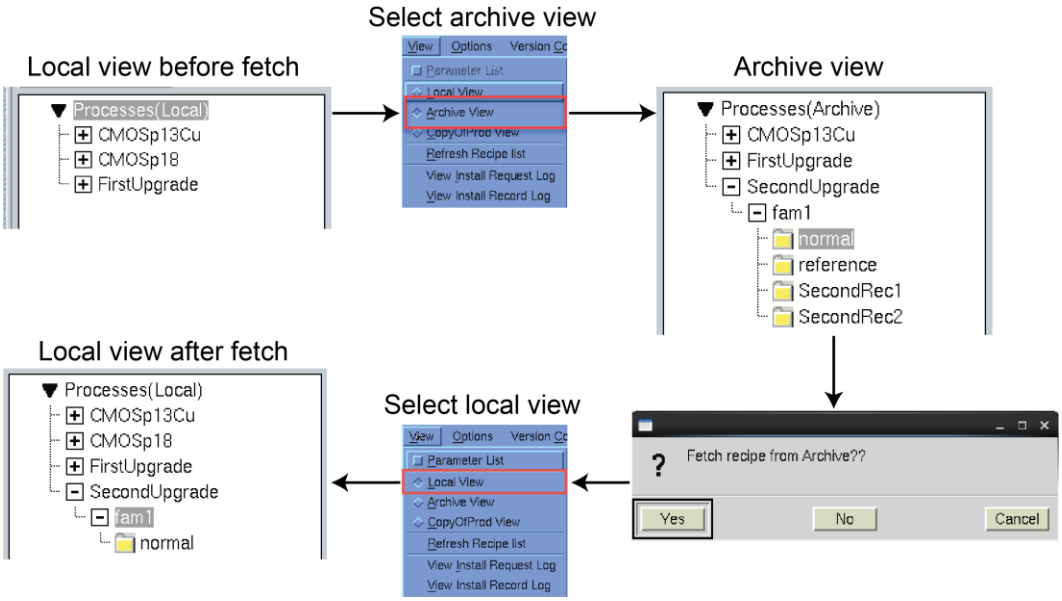
Figure 136: Fetch recipe from Archive dialog box



3. Select **Yes** to place a read-only copy of the recipe in the local directory.
4. Select **View > Local View** to return to the local view of the recipe navigator.

[6] Or a corresponding file. For example, when you right-click a test macro name to open the Test Script Editor (TSE) tool, a test structure file (.tsf) file opens when the tools starts.

Figure 137: Adding a recipe from the archive to the local directory



The recipe is added to the recipe navigator under the appropriate process and product family. If the appropriate process and product family were not previously displayed in the recipe navigator, these are also added.

NOTE

Use the above method to populate a new local area, the directories of which are initially empty. Adding new recipes automatically places the recipes and their supporting files into the local area so you can then modify them.

Revising a recipe

You can revise recipes (`.krf` files) directly in Keithley Recipe Manager (KRM).

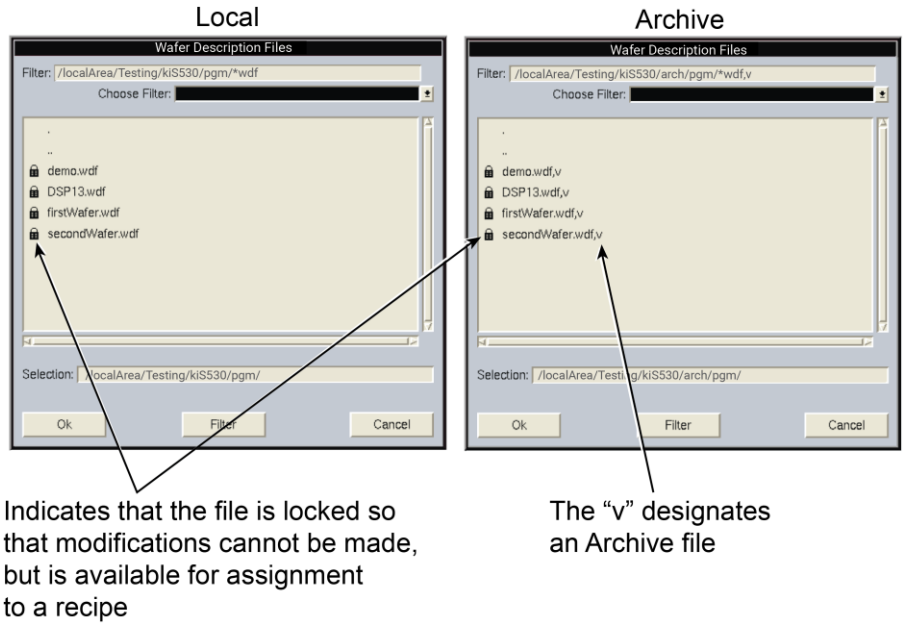
To revise a recipe:

1. In the recipe navigator, select the recipe you want to modify.
2. Bring a writable copy of the recipe into the local area by selecting one of the following:
   - **Version Control > Other Operations > Check Out**
   - **Version Control > Modify**
   - Right-click the recipe in recipe navigator and select **Modify**
3. Make changes on the Recipe Contents and Parameters tabs as required. You can change the supporting files by selecting the yellow folders labeled L (local area) or A (archive) where present, or by direct editing, where allowed.

NOTE

The availability of file selection directories and direct editing in user-defined fields is governed by the `krm.ini` file. See Configuring user-defined recipe contents fields and terminology (on page 7-89) for more information.

Figure 138: Differences between files in local view versus archive view



4. When you finish making changes, select **File > Save** to save the modified recipe in your local area.
5. If you are satisfied that the recipe is ready to be assigned a new revision number, select **Version Control > Other Operations > Check In**. If you want to discard the changes, select **Version Control > Other Operations > Revert**.

   For more information about checking in files, see Checking files into the archive (on page 7-58); for more information about reverting files, see Reverting files to precheck-in status (on page 7-59).

Revising supporting files and user libraries

Revise supporting files using Keithley Test Environment (KTE) tools through Keithley Recipe Manager (KRM).

To revise a supporting file:

1. Right-click the supporting file name on the KRM Recipe Contents or Parameters tab to open a shortcut menu.
2. Select the appropriate KTE tool (Modify option) for the file on the shortcut menu (see Using Keithley Recipe Manager with KTE tools (on page 7-31)). The selected tool opens.

NOTE

Not all supporting files and tools can be opened directly from KRM. Some secondary tools and supporting files must be opened indirectly through menus and other selections in the user interface of the primary tools that you can open through KRM.

When you right-click a file on the Recipe Contents or Parameters tab to start a tool, the file automatically opens when the tool starts. For example, when you start the Keithley Interactive Test Tool (KITT) by right-clicking and selecting KITT `c13cu_modc.ktm` on the **Parameters** tab and selecting **KITT (Modify)**, the `c13cu_modc.ktm` macro is displayed in the KITT Test Macro Editor.

3. Select the supporting file that you want to revise and select **Version Control > Other Operations > Check out** to check out the supporting file.
4. Modify the supporting file in the appropriate KTE tool (for more information about KTE tools, see the Software (on page 6-1) section of this manual.
5. Save the modified supporting file.
6. If the supporting file is ready to be assigned new revision number, select **Version Control > Other Operations > Check In** to check the file into the archive. If you want to discard the changes, select **Version Control > Other Operations > Revert** to revert the supporting file to its former status. For more information, see Checking files into the archive (on page 7-58) and Reverting files to precheck-out status (on page 7-59).

Checking files into the archive

After you have revised or created a new recipe or supporting file, check it into the archive to make it available for use in production.

To check a recipe file into the archive:

1. In recipe manager, select the recipe file.
2. Select **Version Control > Other Operations > Check In**.
3. Add comments and set the revision number as necessary and select **OK**.

Checking in a newly created file places an unlocked version of the file in the archive directory, establishes it as the head (production) version, and assigns it head version 1.1.

Checking in a revised file (that you checked out before revising) places it in the archive directory, removes the lock, and labels it as head version with the next sequential revision number.

NOTE

You can check a file into the archive for the first time with a revision number offset. For more information, see Start with a revision offset on initial entry into source control (on page 7-58).

Start with a revision offset on initial entry into source control

You can check a file into the archive for the first time with a revision number offset. This is useful when you want to use identical limits files with the same revision number on different model systems.

To set a revision offset on a file on first load to the archive:

1. In the Keithley Test Environment (KTE) tool you are using, select **Version Control > Other Operations > Check In**. The Check In dialog box is displayed.
2. Select the **Set Revision** button.
3. Enter the revision number you want in the dialog box and select **OK**.
4. Select **OK** in the Check In dialog box.

Keithley Recipe Manager (KRM) applies this revision number to the file upon completion of the Check In process.

NOTE

The check-in process fails if the revision number entered is not higher than the current version in the archive.

If you pass the `$KIBIN/checkInLabel` routine the `-r` switch, the supplied argument is used as the initial revision number. You can then use a script to call the `checkInLabel` routine with the appropriate starting revision number for a file. This allows you to populate the archive with files from another facility that have identical revision numbers upon check in.

Reverting files to precheck-out status

If you check out a file and then make no changes or want to discard changes you made, you can revert it to the state it was in before you checked it out by selecting **Version Control > Other Operations > Revert**. The lock is removed in the archive and changes in the local area are discarded.

Releasing and installing a recipe or supporting file

When a recipe or supporting file is ready for production, a Release operation in Keithley Recipe Manager (KRM) sets a PROD label on the files. The Install operation bundles the waiting recipes and individual supporting files in the archive and puts them in staging directories. The staging directories for clients are located on the server, which also contains the archive. This ensures that KRM can update all staging areas when requested, regardless of client activity.

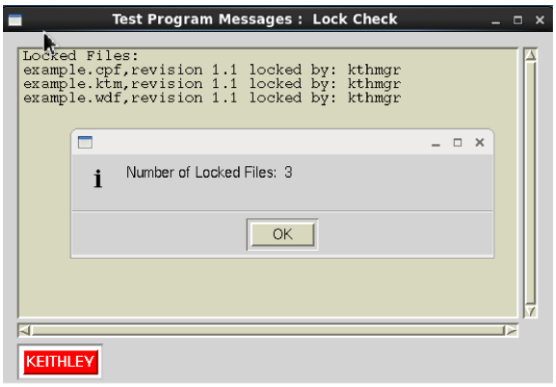
NOTE

If a recipe has been released, but not yet installed, the release operation can be reversed by selecting **Version Control > Other Operations > Remove Label**. This removes the PROD label. If the files have been installed, you cannot reverse the release and installation.

To release and install a recipe:

1. In KRM, select the recipe that you want to release and install.
2. Select **Version Control > Other Operations > Release**. A Validate? dialog box is displayed.
3. Select **Yes** to validate the recipe or **No** to skip validation if you have already validated the recipe. A Check for locked files? dialog box is displayed.
4. Select **Yes**. A lock-check message and number-of-files dialog box is displayed.

Figure 139: Checking for locked files



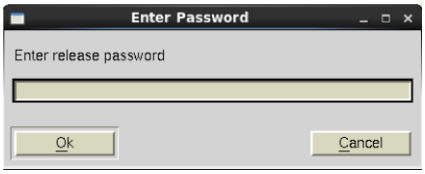
NOTE

The supporting files listed in the Test Program Messages: Lock Check window are checked out and may be in the process of revision. If the recipe you are releasing and installing uses a listed supporting file, you may want to delay release of the recipe until the revised file is released.

5. In the Number of Locked files dialog box, Select **OK**. A Release File / Set Label window is displayed.

Figure 140: Recipe Release File / Set Label dialog box

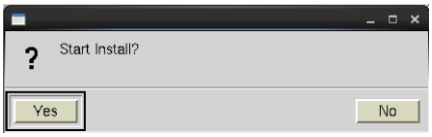
6. Select the options you want and select **OK**. The Enter Password dialog box is displayed.



7. Enter the release password (default release password is `kthadm`).

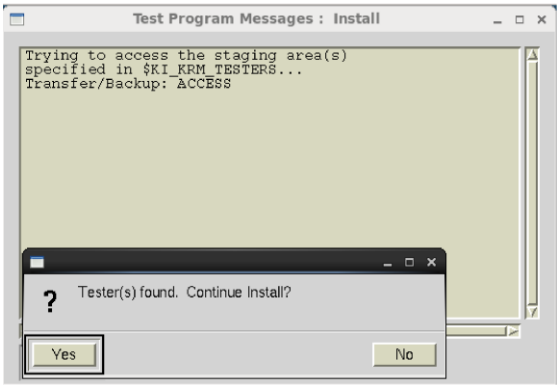
8. Select **OK**. A Start Install? dialog box is displayed.

Figure 141: Start Install? dialog box



9. Select **OK**. A dialog box stating that clients have been found and asking if you want to continue the install action.

Figure 142: Testers found dialog box

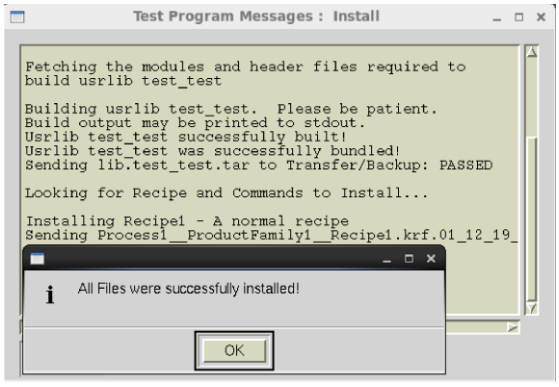


NOTE

KRM determines which clients to update using a list of clients created as part of the KRM set up process.

10. Select **Yes** to continue the installation. When the installation is complete, a dialog box is displayed that says `Successfully installed`.

Figure 143: Install log and successfully installed dialog box



11. Select **OK** to complete the installation.

Alternate production-file installation procedure

You can set up Keithley Recipe Manager (KRM) to run a script when you select **Version Control > Install** that updates the version of any recipe whose supporting files have been changed. The `$KIBIN/install_files.pl` script is identified by the `KI_ALT_KRM_INSTALL` environment variable.

NOTE

Before running this script, make sure the following steps have been completed (these steps usually occur during KRM setup):

- Execute the `$KIBIN/mkArcSubdirs` command to create the `$KI_ARCHIVE/recipe/RCS` directory. This directory is the location of the `<recipe_name>.adm` archive.
- Create the `$KI_ARCHIVE/install_recipe.ini` file and the temporary staging directory using one of the following methods:
  – Execute the `$KIBIN/vc_add_tester_2_server` script.
  – Create the file directory manually, using the `$KIBIN/vc_add_tester_2_server` script as a reference.

To set up the alternate production-file installation:

Edit the `$KIHOME/.ki_setup_option_5_vc` file to include `setenv KI_ALT_KRM_INSTALL $KIBIN/install_files.pl`.

The `install_files.pl` script processes the files in the following steps:

1. Makes a copy of the `InstallReq.log` file.
2. Sets the `KI_KRM_TESTERS` environment variable to the `install_recipe.ini` file. This file specifies placement of bundles into a temporary staging directory.
3. Executes `krm -install`, which places bundles in the temporary staging directory using the normal KRM installation process.
4. Updates the recipe versions as follows:
   - Scans the `$KI_ARCHIVE/CopyOfProd` area to determine which recipes use any of the Keithley Test Environment (KTE) files that were modified and creates a list of these recipes.
   - Uses `docTreeList` to generate a list of files and the revision used by the recipe. The `<recipe_name>.adm` file is created or modified to contain the list of files and revisions (where `<recipe_name>` is the name of the recipe).

NOTE

You can view <recipe_name>.adm file revisions by selecting the recipe in KRM and selecting **Version Control > Recipe.adm Operations**. This selection is only available if a <recipe_name>.adm file exists. This allows you to see the history of the file, compare two revisions of the file, or view the contents of the file.

5. Checks the <recipe_name>.adm file into the archive and also places the file in a temporary staging directory.
6. Copies the bundles and .adm files into the normal staging directories.
7. Cleans up the temporary staging directory.

Keithley Recipe Distributor (KRD) and Keithley Test Execution Engine (KTXE) place the .adm files in the $KI_KRM_NORM/plans($KI_KTXE_KRF) directory.

KTXE reads and extracts the revision number of the <recipe_name>.adm file and places the value into the datapool as a CHAR_P value named ki_recipe_version.

If KTXE cannot find a <recipe_name>.adm file, it uses the revision number of the <recipe_name>.krf file. The datapool value can be referenced and reported as the recipe version number to your data collection system at UAP_PROG_ARGS or later.

NOTE

You can use another script instead of install_files.pl to provide additional capabilities or different behavior than described above.

Moving recipes between sites

Use the Keithley Component Manager (KCM) utility to move recipes between sites, as described in the following topics. For more information about KCM, refer to Keithley Component Manager (KCM) utility description (on page 6-163).

Importing a recipe

To unbundle a <RecipeFilename>.tar file and distribute the files into the appropriate directories of the presently active project, execute the $KIBIN/kcm -i <RecipeFilename> [-u] script.

Where:

- -i specifies the kcm import function
- -u specifies inclusion of the user libraries

The utility reports the progress of this operation.

Exporting a recipe

To create a `<RecipeFilename>.tar` file (bundle) that includes the recipe, all required KTE files, and all required user libraries (optional), execute the `$KIBIN/kcm -e <RecipeFilename> [-u]` script.

Where:

- `-e` specifies the KCM export function
- `-u` specifies inclusion of the user libraries

The utility reports the progress of this operation.

Deleting items from the local area

The following topics describe the tasks necessary to delete a recipe and related files from the local area.

Deleting a recipe from the local area

NOTE

Do not delete a checked-out recipe or supporting file. After a checked out file is deleted, you cannot directly remove its archive file lock.

If you unintentionally delete a checked out file, you can remove its lock by doing a **Version Control > Fetch operation** on the file that you deleted, and then selecting **Version Control > Revert** to revert the file. After you have done this, you can safely delete the file from your local area.

To delete a recipe:

1. Make sure the KRM Recipe Navigator is in local view (**View > Local View**).
2. In the Recipe Navigator, select the recipe that you want to delete.
3. Select **File > Delete**. A confirmation dialog box is displayed.
4. Select **Yes** in the confirmation dialog box.

Deleting supporting files from the local area

NOTE

Do not delete a checked-out recipe or supporting file. After a checked out file is deleted, you cannot directly remove its archive file lock.

If you unintentionally delete a checked out file, you can remove its lock by doing a **Version Control > Fetch operation** on the file that you deleted, and then selecting **Version Control > Revert** to revert the file. After you have done this, you can safely delete the file from your local area.

To delete a supporting file from the local area:

1. Start the Keithley Test Environment (KTE) tool that is used to edit the type of file that you want to delete. See Accessing KTE tools through Keithley Recipe Manager to see how to access these tools from KRM.
2. In the KTE tool, select **File > Delete**. If a file opens when you open the tool from a KRM shortcut menu, selecting **File > Delete** does not automatically delete the file that opens.
3. Select the file to be deleted and select **OK**. A confirmation dialog box is displayed.
4. Select **Yes**.

Deleting a product family from the local area

To delete a product family:

1. Make sure the Keithley Recipe Manager (KRM) recipe navigator is in local view by selecting **View > Local View**.
2. In the recipe navigator, delete all recipes in the product family (see Deleting a recipe from the local area (on page 7-64) for instructions).
3. Select **View > Archive View** to switch KRM to archive view.
4. Select **View > Local View to** switch KRM back to local view. The product family no longer appears.

Deleting a process from the local area

To delete a process from the local area:

1. Select **View > Local View** to make sure the Keithley Recipe Manager (KRM) recipe navigator is in local view.
2. In the recipe navigator, under the process you want to delete, delete all recipes in all product families (see Deleting a recipe from the local area (on page 7-64) and Deleting a product family from the local area (on page 7-65)).
3. Select **View > Archive View** to switch KRM to archive view
4. Select **View > Local View** to switch KRM back to local view. The process and all product families below it no longer appear.

Using batch operations to do common tasks in KRM

The Batch menu allows you to perform certain common operations to multiple marked product families or recipes at the same time. Batch operations are only available in the local view.

Batch operation phases

When a batch operation is initiated, a check phase is started and all marked recipes are tested for required conditions such as being archived or writable. All marked or potential target files are checked and any recipes that do not pass all the checks are displayed in the Check window.

At the end of the check phase, a list of recipes that passed the checks is displayed. Batch operations will be performed on these files. You will be prompted to continue the operations; select **Yes**, **No**, or **Cancel**.

The progress of the tests is displayed in the Run window. Recipes are listed in the window with pass or fail information as they are processed. A failure may or may not abort the batch operation, depending on the type of failure.

To see more detailed failure information, look in the `$KILOG/krm.log` file.

Batch marking

Though only a single file tree node can be selected at any one time (grayed text), multiple nodes can be marked with a blue icon at the same time.

A node on the file tree is selected (grayed text) by clicking on the name of the node (not its icon).

A node on the file tree is marked for batch processing by holding down the Ctrl key while you click the icon for the node (not its name). Ctrl-clicking will mark the node, turning its icon blue. Icons nested under the marked node may also turn blue, depending on the current selection.

If no recipe is selected when a product family node is marked, all recipes under the node are also marked. If a recipe is selected when a product family node is marked, only a recipe with the same name as the selected recipe will be marked. If there are no recipes with the same name in the product family, no recipes are marked.

If you want to mark all recipes under a node, make sure that a process or product family node (not a recipe) is selected.

If the Save Copies To operation is initiated on a marked product family, the newly created recipe is marked.

Batch icon marking and level indicators

A blue icon indicates that node, or the node above it was Ctrl-clicked when unmarked. Marking indicates that the node may be a target of a batch operation.

A nonrecipe icon may have a small level indicator bar beside it. These indicate the following:

- **No bar:** There are no marked icons under this icon level.
- **Empty:** There are marked product families, but no marked recipes under this icon level.
- **Half:** There are some marked recipes under this icon level.
- **Full:** All recipes under this icon level are marked.

Save Copies To batch operation

Copies the selected recipe to all marked families, and updates the Name and Time fields. If the selected recipe was archived, then it archives the new recipes. If the head label of the selected recipe was PROD, the new recipes are assigned a PROD label and are added to the Install Request log. If a recipe with the same name already exists in any of the marked families in the local area or in the archive, no copy is made.

The Save Copies To operation will not copy over a marked recipe; it will only create new copies in the marked families. The file name and timestamp fields of the new recipes are updated.

When the operation completes successfully, the new copies in each family are marked, readying them for other possible batch operations.

Prerequisites for the Save Copies To operation:

- The new recipe names must not exist in the local directory.
- The new recipe names must not be archived.

Search/Replace batch operation

If recipes are marked and an item on the Recipe Contents tab is highlighted, the contents of the highlighted edit field are written to all marked recipes.

To select an item to replace, select the edit field on the Recipe Contents tab. If replacement is allowed, a dark order is visible around the edit field. This border must be present when the Search/Replace operation is selected in the Batch menu. The data in the edit field may belong to any selected recipe, and does not need to be editable.

NOTE

The label on the field may not be the same as the tag in the Keithley Recipe Manager file. For example, the Cassette Plan field in the editor corresponds to the `.cpf` fields in recipe files.

Prerequisites for the Search/Replace batch operation:

- The search and replace strings must be defined.
- The recipes must exist in the local directory.
- The recipes must be writable.

Search/Replace/Release batch operation

If recipes are marked and an item on the Recipe Contents tab is highlighted, the contents of the highlighted edit field are written to all marked recipes. As required, the recipes may be checked out, altered, checked back in, and released. This applies a PROD label to the recipes and adds them to the install request queue.

The end result is that all marked recipes are modified, checked-in, unlocked, labeled, and released.

CAUTION

This option must be used carefully; it is enabled and disabled using the `enableSrchRepRel` key in the `krm.ini` file. A release password is required for this option.

Prerequisites for the Search/Replace/Release operation:

- The search and replace strings must be defined.
- The recipes must exist in the local directory.
- If a recipe is locked, that lock must be held by the user performing the operation.

Check Out batch operation

All marked recipes are checked out and locked when you select this operation.

Prerequisites for the Check Out batch operation:

- The recipes must exist in the local directory.
- The recipes must be archived.
- The recipes must not be writable.
- The recipes must not be locked.

If any of these prerequisites are not met, the operation is aborted and no files are checked out.

Check In batch operation

The Check In batch operation checks in in all marked recipes to the archive with a common log comment. Files are checked in to the latest revision with no labels.

Prerequisites for the Check In batch operation:

- The recipes must exist in the local directory.
- The recipes must be read-write.
- The recipes must be checked out by the user performing the batch operation, or they must not be in the archive.
- If a recipe is locked, the lock must be held by the user performing the batch operation.

Label batch operation

This batch operation labels the head version of all marked files that have been archived with the specified label. If there is a labeling conflict, the existing label is moved to the head revision.

Prerequisites for the Label batch operation:

- The recipes must exist in the local directory
- The recipes must be archived.

Remove Label batch operation

This batch operation removes the labels of all marked files with a specified label.

Prerequisites for the Remove Label operation:

- The recipes must exist in the local directory.
- The recipes must be archived.

Release batch operation

This batch operations labels all marked files with a PROD label and adds them to the `InstallReq` log. A release password is required for this operation.

Prerequisites for the Release batch operation:

- The recipes must exist in the local directory.
- The recipes must be archived.
- The latest versions of the recipes must have a PROD label.

UnRelease batch operation

This batch operation removes the PROD label from all marked files and adds a DELETE flag in the `InstallReq` log. A release password is required for this operation.

Prerequisites for the UnRelease batch operation:

- The recipes must exist in the local directory.
- The recipes must be archived.
- The most recent version of the recipes must be labeled PROD.

Delete batch operation

The batch operation deletes all marked recipes from the local directory. If the recipes are not archived, this operation is not reversible. A release password is required for this operation.

Prerequisites for the Delete batch operation:

- The recipes must exist in the local directory.
- The recipes must not be locked in the archive by the user performing the Delete operation.

UnMark All batch operation

This batch operation unmarks all marked nodes, including hidden nodes, in a single operation.

Examples

The following topics contain examples of Keithley Recipe Manager (KRM) applications. Each example builds upon the example before it to show the whole development process.

Example 1: Create a new recipe using existing files

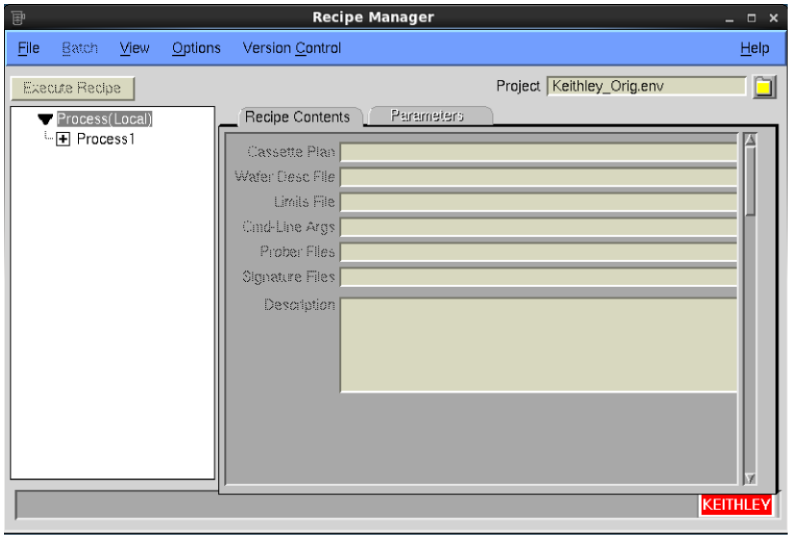
This example leads you through typical recipe-creation tasks in the Keithley Recipe Manager (KRM) engineering mode, from naming the recipe to validation and testing. The example includes the following actions:

- Creating a new recipe in the local area.
- Changing the limits file (`.klf`) for the new recipe.
- Checking the new recipe into the archive.
- Validating and testing the new recipe.

To create the new recipe:

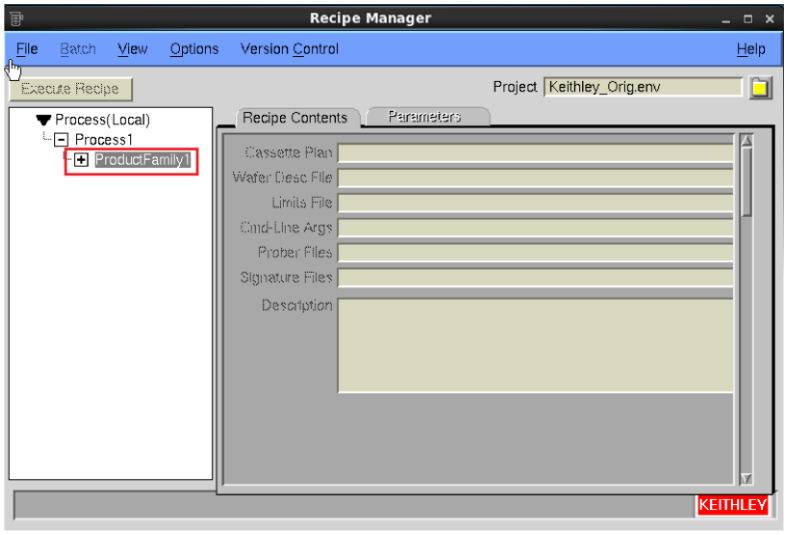
1. Right-click in an unpopulated area of the screen. A shortcut menu is displayed.
2. Select **Open in Terminal**.
3. At the command prompt, enter `krm&` to start KRM in engineering mode.
4. The KRM interface opens.

Figure 144: Keithley Recipe Manager user interface



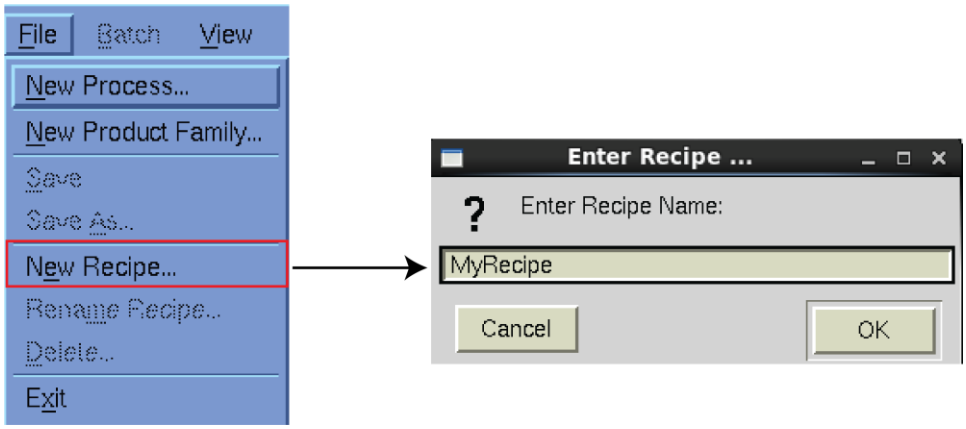
5. In the recipe navigator on the left side of the screen, select a product family. This is where you will create a new recipe.

Figure 145: Select existing product family



6. In the File menu, select **New Recipe**. The Enter Recipe dialog box is displayed.

Figure 146: Create a new recipe



7. Enter `myRecipe` as the new recipe name and select **OK**. See Creating a new recipe (on page 7-42) for more specific information on recipe naming requirements.
8. Select **OK**. The new recipe is visible under the product family you selected.

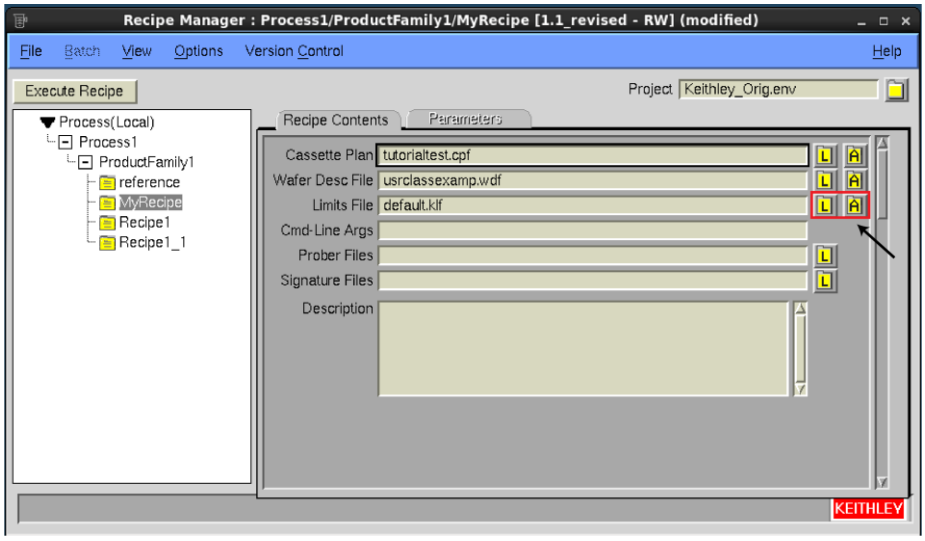
Figure 147: Results of adding a new recipe

- In the recipe navigator, the new recipe is added to the displayed list of recipes, and the new recipe is selected.
- In the Recipe Contents tab, a cassette plan, a wafer description file, a limits file, and a description are displayed. These are already assigned to the new recipe, which is a copy of the reference recipe for the product family that the new recipe is part of.

NOTE

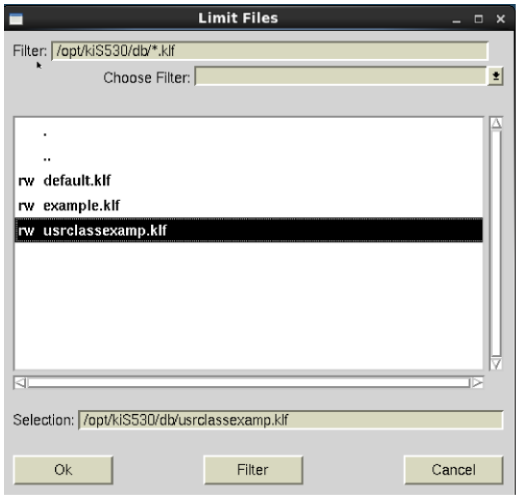
If your recipe has no files shown on the Recipe Contents tab, there may not be a reference recipe for the product family where you created the new recipe. If this is true, you can select the supporting files using the following instructions. To create a reference recipe, see Creating a new product family under an existing process (on page 7-41).Select a yellow folder to the right of the Limits File box. You can select a file from the local area (L folder) or the archive (A folder).

Figure 148: Selecting a limits file



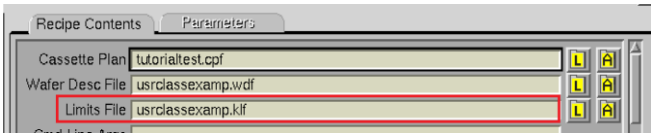
9. Select a different `.klf` file than was originally displayed in the Limits File box (ignore padlock-shaped symbols that may be next to some of the filenames). For this example, `usrclassexamp.klf` is selected.

Figure 149: Available limits files



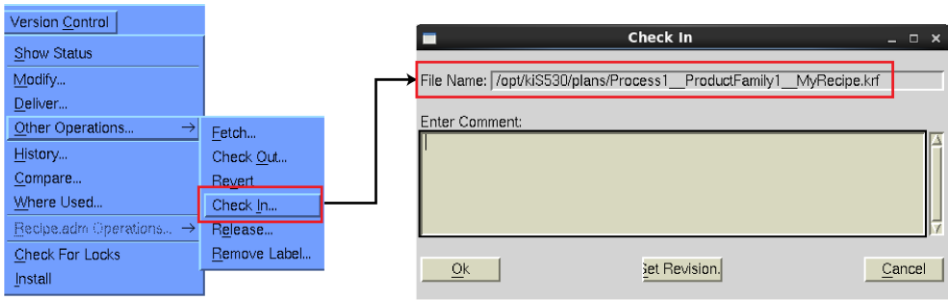
10. Select **OK**. In the Recipe Contents tab, the Limits File box shows the name of the new limits file.

Figure 150: Existing limits file in new recipe



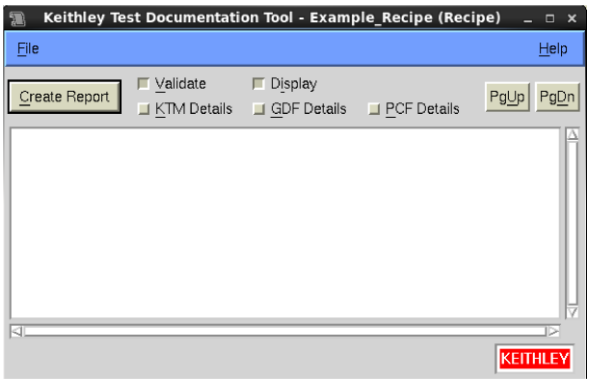
11. Select **File > Save** to save the new recipe to the local area.
12. Select the **Version Control** menu and select **Other Operations**.
13. Select **Check In**. The Check In window opens, displaying the name and path of the new recipe in the local area.

Figure 151: Checking in the recipe



14. In the Check In window, enter a comment to record relevant information about the new recipe.
15. Select **OK**. The Check In window closes.
16. With the new recipe still selected in the recipe navigator, select **Options > Validate Recipe**. The Keithley Test Document Tool (KTDT) opens.

Figure 152: Keithley Test Documentation Tool

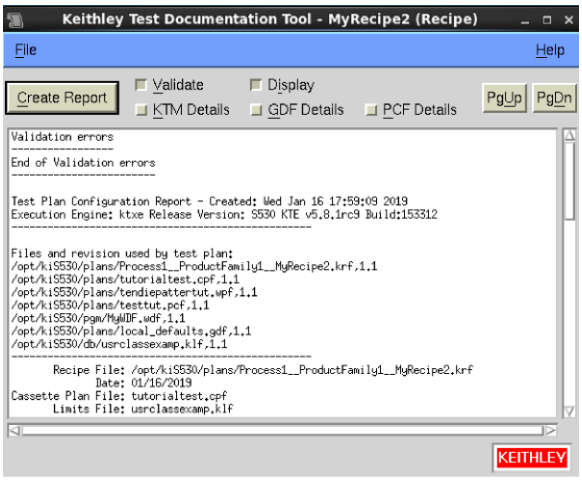


17. Select **Create Report**. The following sequence of screens is displayed.

Figure 153: KTXE execution message

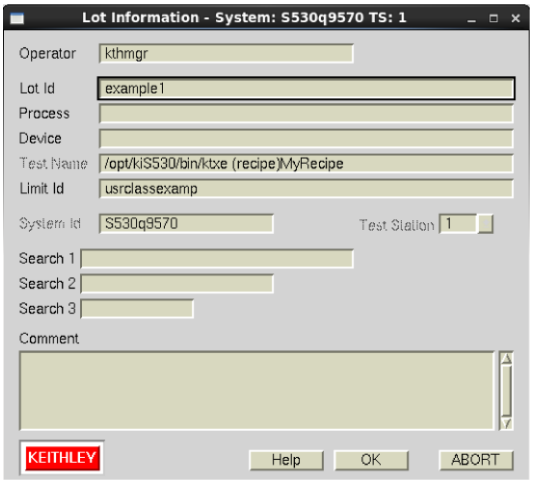
Figure 154: Building Report KTDT message

Figure 155: Recipe validation report



18. Select **File > Exit** to close the Keithley Test Documentation Tool.
19. Test the validated recipe by selecting the new recipe and selecting the **Execute Recipe** button in the upper left corner of the KRM interface. A Lot Information dialog box is displayed.

Figure 156: Execute Recipe Lot Information dialog box



20. Enter the Lot Id identifier and select **OK**. The Keithley Test Execution Engine (KTXE) runs. For more information about KTXE, see the Software (on page 6-1) section of this manual).

NOTE

When the test run concludes, if you plan to continue with example 2, do not delete the recipe that you created in this example. The recipe you created is the starting point for example 2.

Example 2: New recipe, wafer description file from existing files

This example guides you through the following:

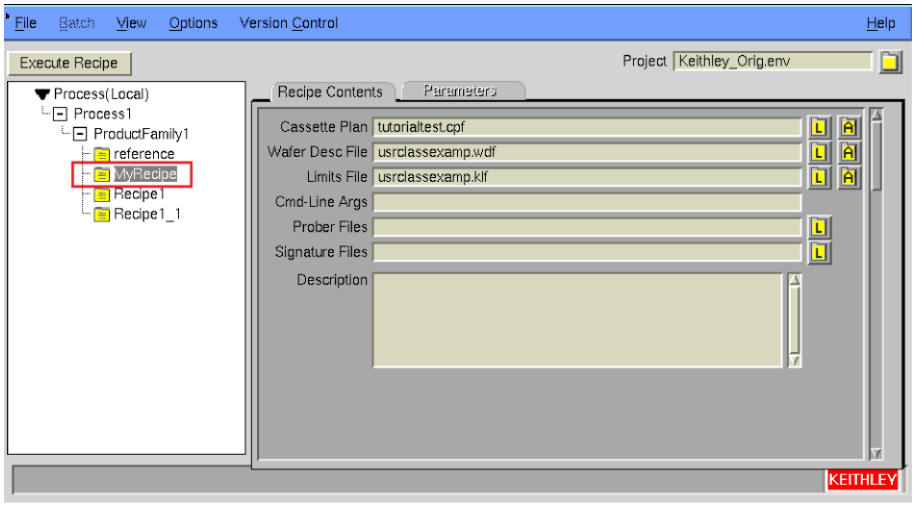
- Creating a new `.wdf` file by saving a `.wdf` file in an existing recipe under a new name.
- Modifying the new `.wdf` file.
- Checking the new `.wdf` file into the archive and then releasing the file.
- Creating a new recipe by renaming the existing recipe and replacing the original `.wdf` file with the new `.wdf` file.
- Checking the new recipe into the archive.
- Validating and testing the new recipe using the procedures described in example 1.

For illustration purposes, this example starts with the `MyRecipe` recipe that was created in example 1. We recommend that you complete example 1 before starting example 2 (refer to Example 1: Create a new recipe using existing files (on page 7-71)).

To create a new recipe and new wafer description file form existing files:

1. In Keithley Recipe Manager (KRM), select the recipe that you created in example 1 (`MyRecipe`).

Figure 157: Selecting an existing recipe

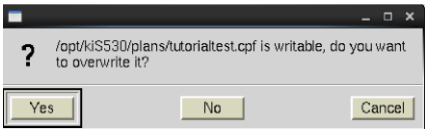


2. Select **Options > Load Recipe**. This loads the supporting files for the recipe into the local area.

NOTE

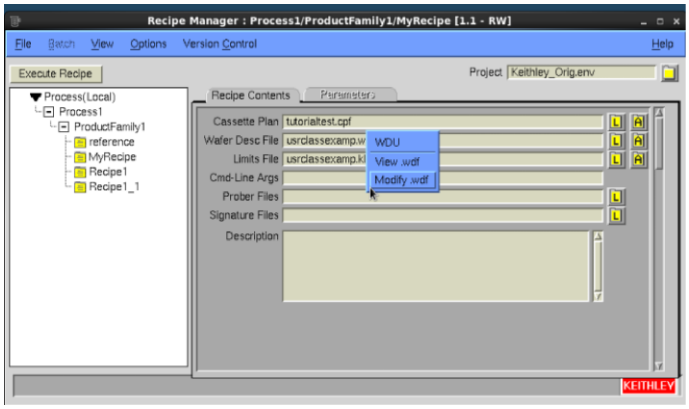
One or more dialog boxes similar to the following figure are displayed if one or more supporting files are already checked out into the local area. A checked-out file may be in the process of revision. Selecting **Yes** causes the local-area supporting file to be overwritten with the PROD (production-labeled) archive version. Select **No** if you are uncertain whether the file can be safely overwritten. For the purpose of this example, continue to the next step.

Figure 158: Checked-out file dialog box



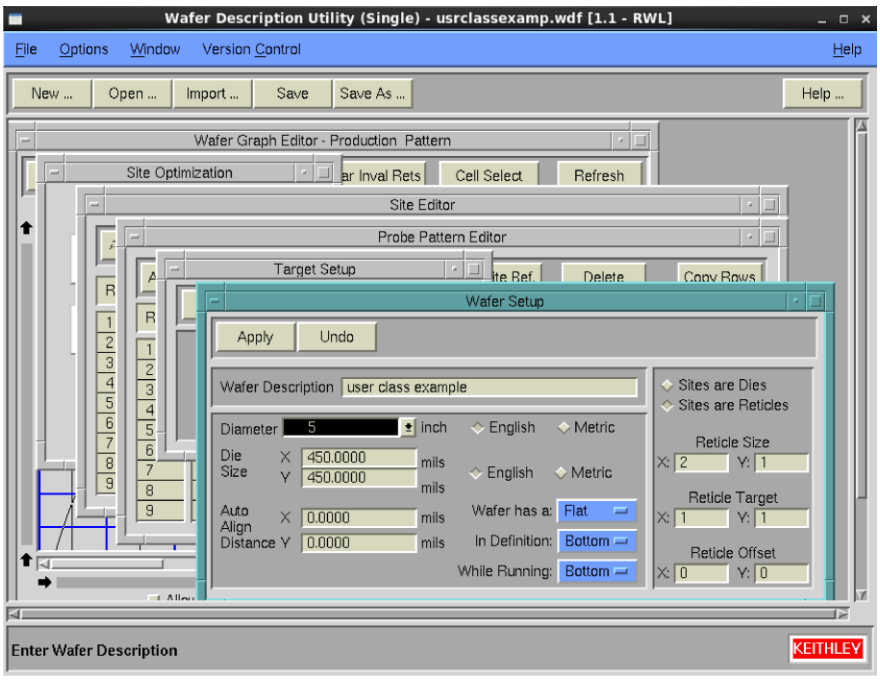
3. Select **Yes** to load the supporting files to the local area. The Recipe successfully loaded!!! dialog box is displayed when all the supporting files are loaded in the local area.
4. Select **OK**.
5. In the KRM Recipe Contents tab, right-click in the Wafer Desc File box. A shortcut menu is displayed.

Figure 159: Accessing the Wafer Description Utility (WDU) through KRM



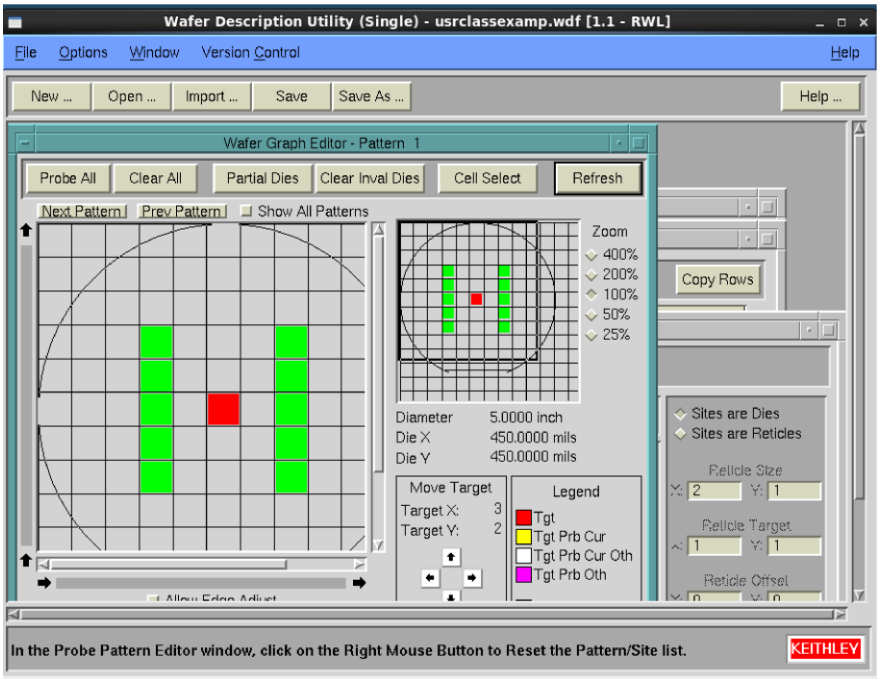
6. Select **Modify .wdf** to start the Wafer Description Utility (WDU), check out the `.wdf` file from archive, and open it.

Figure 160: Wafer description file (.wdf) open in the Wafer Description Utility (WDU)



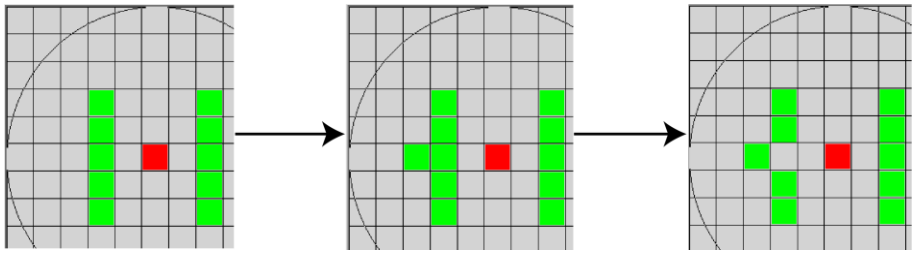
7. Select the Wafer Graph Editor window of the Wafer Description Utility tool.

Figure 161: The Wafer Graph Editor window



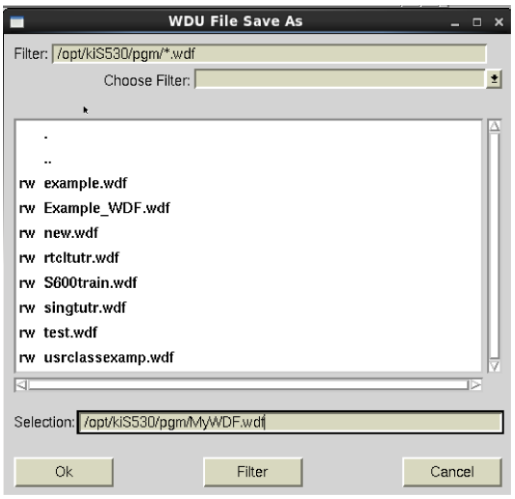
8. Move a probe-pattern site:
   - Locate the middle site in the left column of the probe-pattern sites.
   - Select the site immediately to the left of the middle site in the left column of sites.
   - Select the middle site in the left column of probe-pattern sites to deselect it.

Figure 162: Moving a probe pattern site in the Wafer Description Utility (WDU)



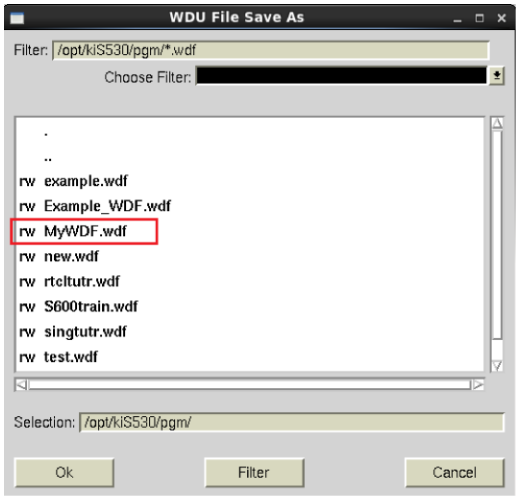
9. Select **File > Save as** in the Wafer Description Utility tool. The WDU File Save As dialog box is displayed.
10. In the Selection box (to the right of the directory path), add `MyWDF.wdf` to the end of the path to rename the `.wdf` file.

Figure 163: Save revised .wdf file with a new name



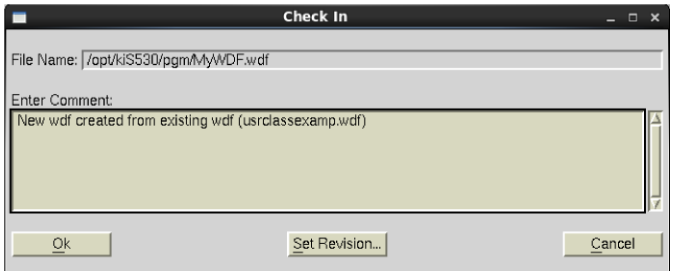
11. Select **OK**. The `MyWDF.wdf` file is added to the available `.wdf` files in the local area. You can verify this by reopening and then canceling the Save As dialog box.

Figure 164: Verifying new .wdf file was created



12. Check the new `.wdf` file into the archive:
    - With the new `.wdf` file open in the Wafer Description Utility, select **Version Control > Other Operations > Check In** to check in the new `.wdf` file. The Check In window is displayed.

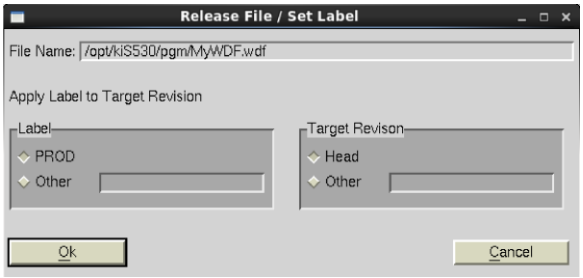
Figure 165: Wafer Description Utility (WDU) Check In dialog box



    - Enter a comment in the Enter Comment box.
    - Select **OK**. The new `MyWDF.wdf` file is checked into the archive.

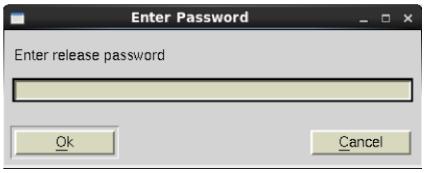
13. In the Wafer Description Utility, select **Version Control > Other Operations > Release**. A `.wdf` Release File/Set Label dialog box is displayed.

Figure 166: Releasing the new .wdf file in the Wafer Description Utility (WDU)

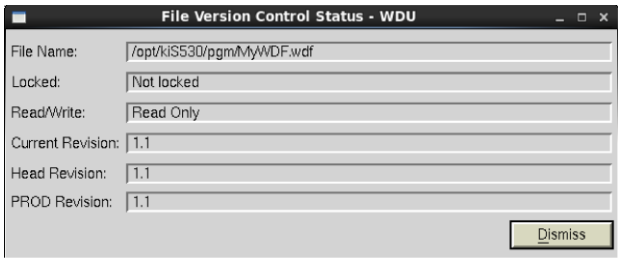


14. Keep the default label settings and select **OK**. The Enter Password dialog box is displayed

Figure 167: Enter the release password



15. Enter the release password in the Enter Password dialog box (`kthadm` is the default password).
16. Select **OK**. The `.wdf` file has now been assigned a PROD label.
17. To verify that a PROD label was assigned to the file, select **Version Control > Show Status**.



18. Select **File > Exit** to close the Wafer Description Utility tool.
19. In KRM, select **File > Save As** to save the new recipe with a different name. The Enter Recipe dialog box is displayed.

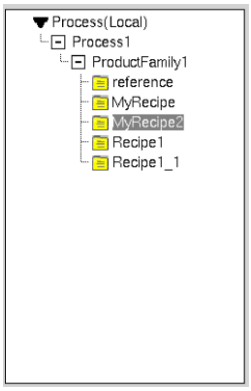
Figure 168: Saving the edited recipe with a new name

20. Enter `MyRecipe2` and select **OK**. The new recipe is visible in the recipe navigator.

NOTE

See Creating a new recipe (on page 7-42) for more specific information on recipe naming requirements.

Figure 169: New recipe visible in the recipe navigator



21. On the KRM Recipe Contents tab, select the yellow L folder to the right of the Wafer Desc File box. The local area Wafer Description Files dialog box is displayed.

Figure 170: Selecting the revised wafer description file

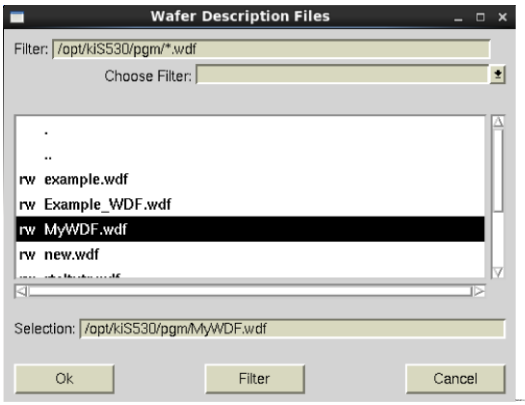
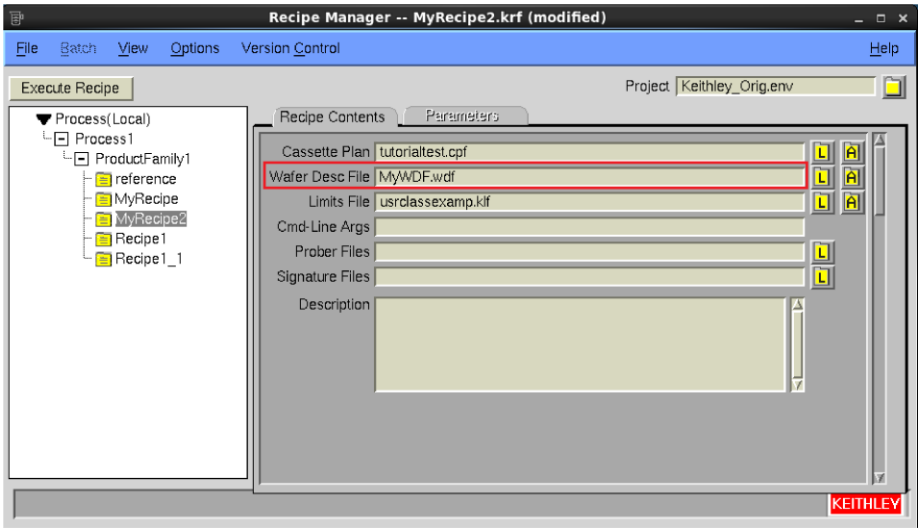
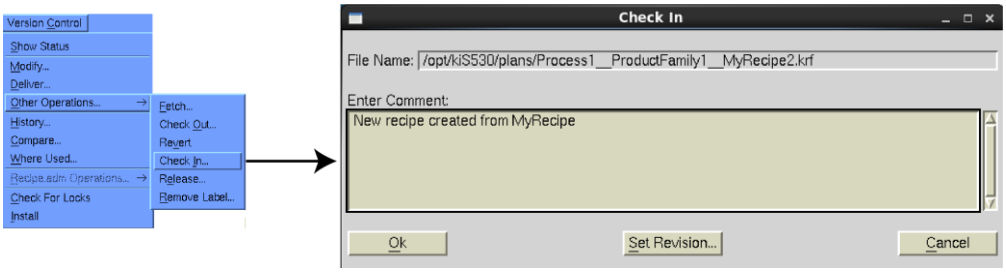


Figure 171: New .wdf file visible on the Recipe Contents tab



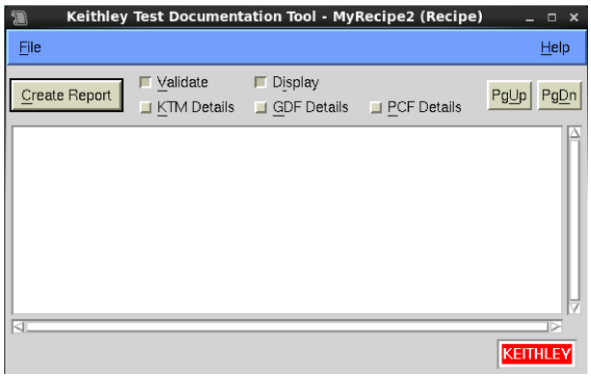
22. Select **Version Control > Other Operations > Check In**. The Check In dialog box is displayed, showing the name and path of the new recipe in the local area.

Figure 172: Checking in the new recipe



23. Enter a comment and select **OK**. The Check In window closes.
24. With the new recipe still selected in the recipe navigator, select **Options > Validate Recipe**. The Keithley Test Document Tool (KTDT) opens.

Figure 173: Validating the new recipe

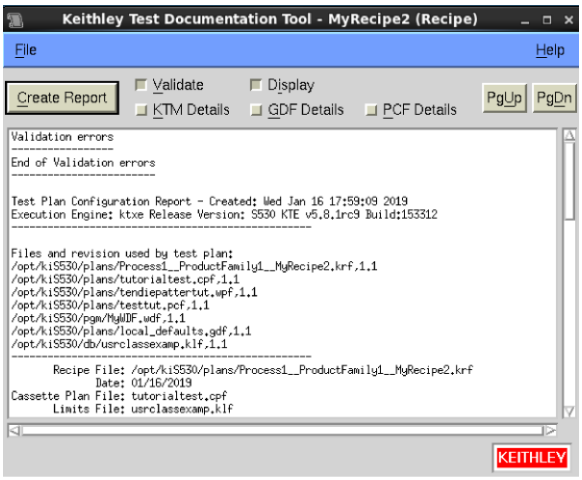


25. Select **Create Report**. The following sequence of screens is displayed.

Figure 174: KTXE execution message

Figure 175: Building Report KTDT message

Figure 176: Recipe validation report



NOTE

If you plan to continue with example 3, do not delete the recipe that you created in this example. The recipe is the starting point for example 3.

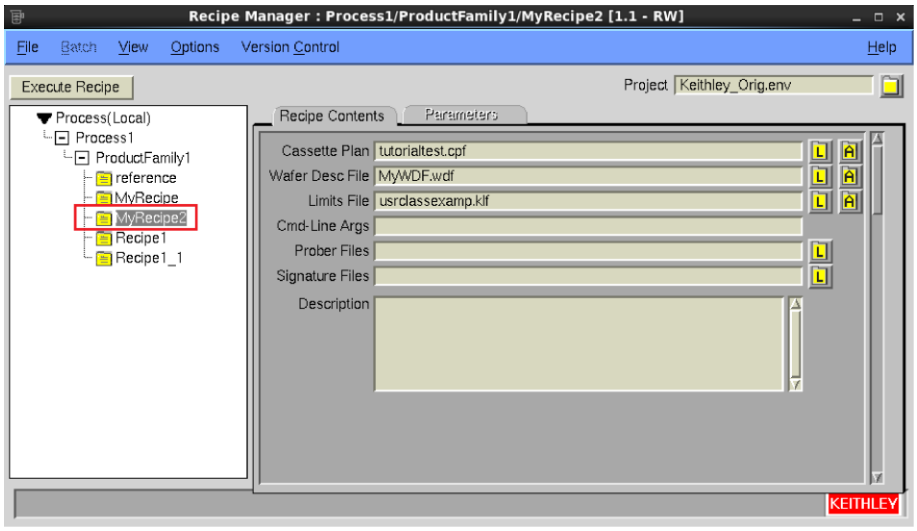
Example 3: Release and install a recipe

This example leads you through the release and installation of a recipe, starting with the `MyRecipe2` recipe that you created in Example 2: New recipe and wafer description file from existing files (on page 7-77).

To release and install a recipe:

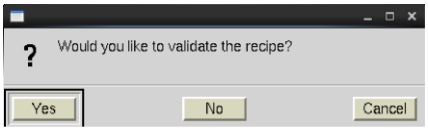
1. In the Keithley Recipe Manager (KRM) recipe navigator, select the recipe you created in example 2 (`MyRecipe2`).

Figure 177: Select MyRecipe2 to release and install it



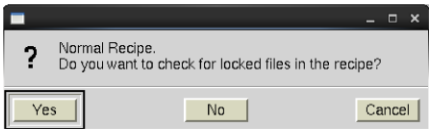
2. Select **Version Control > Other Operations > Release**. A Validate? dialog box is displayed.

Figure 178: Validate recipe prompt



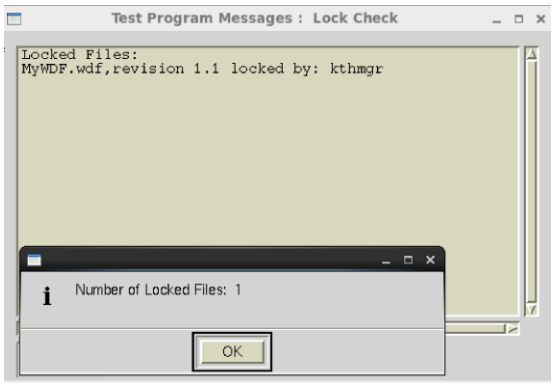
3. Select **No** (you have already validated this recipe in example 2). A Check for locked files? dialog box is displayed.

Figure 179: Check for locked files dialog box



4. Select **Yes**. The Test Program Messages: Lock Check window and Number of Locked Files dialog box are displayed.

Figure 180: Lock Check dialog box showing locked files



NOTE

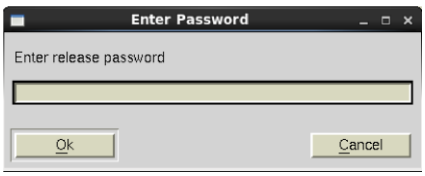
The supporting files listed in the Test Program Messages: Lock Check window are checked out and may be in the process of revision. If the recipe you are releasing and installing uses a listed supporting file, you may want to delay release of the recipe until the revised file is released. For the purposes of this example, continue with the release procedure.

5. In the Number of Locked Files dialog box, select **OK**. A Release File / Set Label window is displayed for the recipe (`.krf` file).

Figure 181: Release File / Set Label dialog box

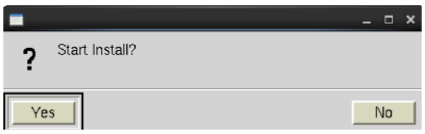
6. In the Release File / Set Label window, accept the defaults and select **OK**. The Enter Password dialog box is displayed.

Figure 182: Enter Password dialog box



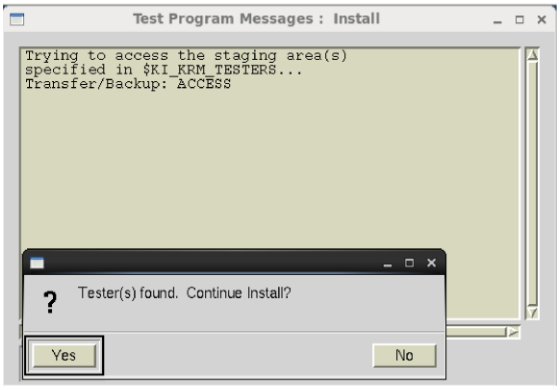
7. Enter your release password and select **OK** (the default release password is `kthadm`). A Start install? dialog box is displayed.

Figure 183: Start Install? dialog box



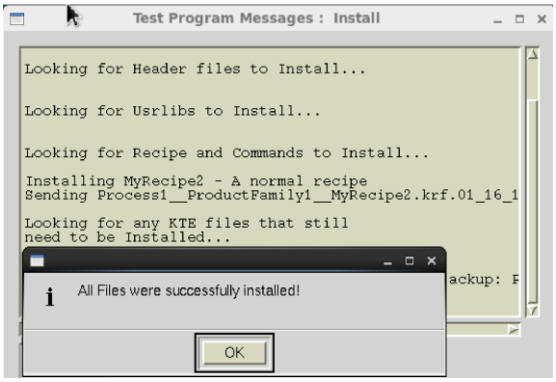
8. Select **Yes**. A Testers Found dialog box is displayed.

Figure 184: Testers found dialog box



9. Select **Yes**. The Test Program: Install window displays the progress of the installation.
10. When the install process has completed successfully, select **OK** in the dialog box that is displayed. The installation concludes.

Figure 185: Test Program: Install window showing all files were successfully installed



Administration

The following topics describe administrative tasks for user accounts and user-defined fields and files. For more in-depth information about version control systems, see Version Control (on page 8-1).

Configuring user-defined recipe contents fields and terminology

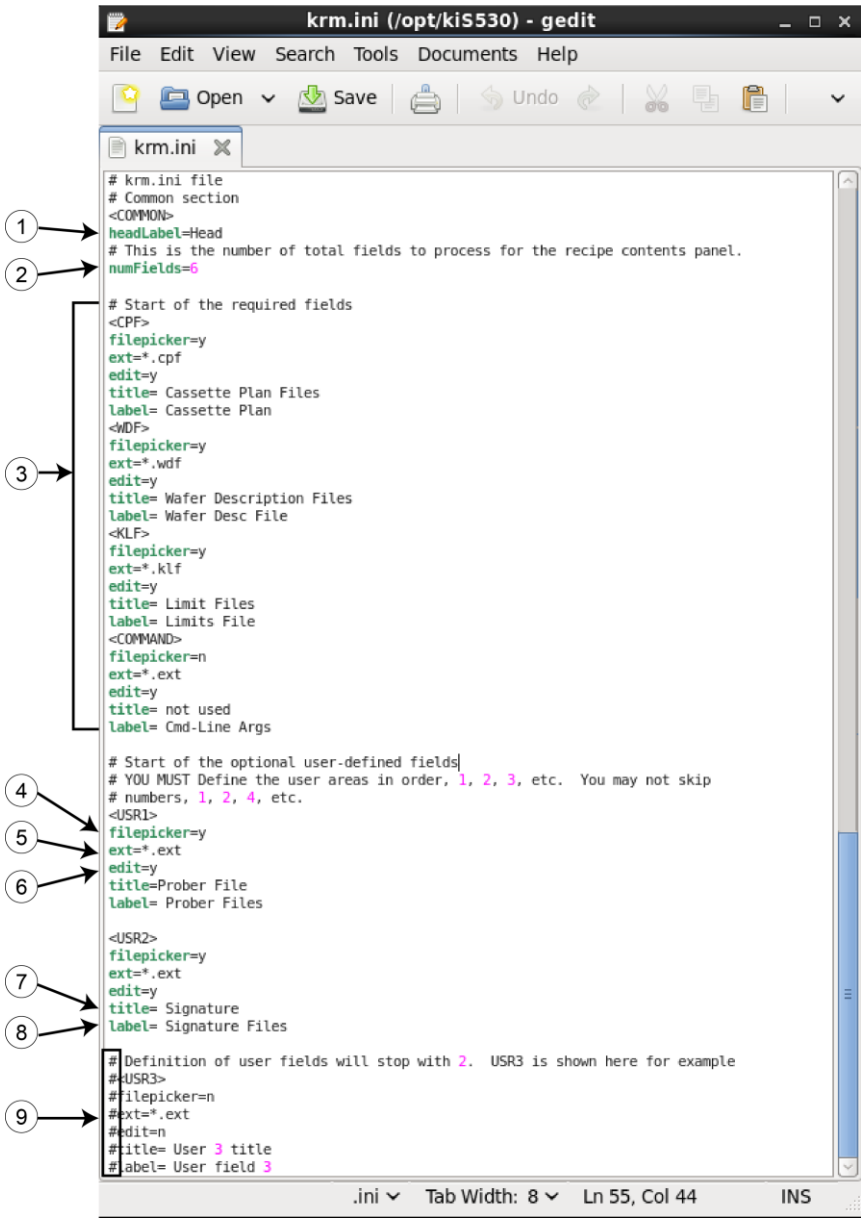
You can configure the `krm.ini` file to define user-defined fields in the Keithley Recipe Manager (KRM) interface and customize the name used for the latest revision of a recipe or supporting file.

You can:

- Set up as many as 32 user-defined fields in the KRM Recipe Contents tab. User-defined fields allow you to input and edit user-specific file information from the KRM window.
- Specify which standard recipe contents fields are viewable and editable. This information is stored as part of the recipe file.
- Change the name that is used throughout KRM for the most recent version of a file. Head is the default designation.

The following figure shows an example of a `krm.ini` file with user-specified fields. See the table after the image for more information about each numbered item.

Figure 186: Example krm.ini file showing settings for user-specified fields



	Entry	Description
1	`headLabel`	Specifies the KRM term for the latest revision of a recipe, supporting KTE file, or supporting user library. The default term in the KRM interface is `Head`.
2	`numFields`	Specifies the maximum number of fields that KRM displays. The value of the number is the sum of the number of required fields and the number of user-defined fields. To restrict the number of fields without adding comments, you can assign a number value that is less than the number of noncommented fields.
3	`required fields`	Do not edit these fields unless you understand the consequences. You can edit the `title`, `label`, `<CPF>`, `<KLF>`, and `<WDF>` entries.
4	`filepicker`	A `y` argument in this entry tells KRM to display a local file selection button (yellow L folder) next to option on the Recipe Contents Tab. For the three required fields, KRM also displays an archive file selection button (yellow A folder). An `n` argument indicates that no file selection button display next to the option.
5	`ext`	Specifies the file extension KRM uses for the default filter in the file selection window. The file selection window displays all files with the specified file extension in the directory from which you start KRM. If no files with that extension are located there, the file selection window displays the subdirectories at that location.
6	`edit`	A `y` argument specifies that the text in the field is directly editable. An `n` argument indicates that the text is not editable.
7	`title`	Specifies the title of the window that is displayed when you select the yellow folder icon next to a field.
8	`label`	Specifies the label to be displayed next to the field.
9	`#`	The # sign before text indicates a comment. By default, all user-defined fields are commented. The first commented user-defined field marks the end of user-defined fields; KRM ignores all succeeding user-defined fields. (Therefore, <USR2> is the last recognized user-defined field in this example.)

CAUTION

Spaces are counted as characters in the `krm.ini` file. Take care to follow the supplied format to avoid errors.

NOTE

The KTE files that are referenced in user-defined fields must be configured elsewhere. For detailed information about setting up these files, see the Software (on page 6-1) section of this manual.

Environment variables for user-defined files in Keithley Recipe Manager

Keithley Recipe Manager (KRM) has 32 user-defined environment variables (`KI_KRM_USER_01` through `KI_KRM_USER_32`). You can use these environment variables to map to optional KRM user-defined fields on the Recipe Contents tab.

You must define these variables in the `$KIHOME/.ki_setup_option_5_vc` file. After defining a variable for a particular user-defined field, the corresponding window in KRM (if enabled) displays the defined value as the default path name.

For more detailed information about environment variables, see the Software (on page 6-1) section of this manual.

Version control in KRM

Files, revisions, and other information for all versions of existing recipes and supporting files (`.cpf`, `.wdf`, and other files and user libraries) are stored in the archive, which is a protected directory on the server (or in a separate archive directory in a single-system configuration).

New recipes and supporting files in development are stored in the local directory until they are ready to be checked in to the archive directory.

The following sequence is an example of the version-control process for revising a recipe in Keithley Recipe Manager (KRM).

1. Check out an existing recipe file from the archive to the local area (a directory reserved for recipe development).
2. Make changes to the checked-out recipe using KRM; revise supporting files by accessing the appropriate Keithley Test Environment (KTE) tool through KRM (for example, the Wafer Description Utility (WDU) or Keithley Test Plan Manager (KTPM)).
3. Update the archive with the revised files using the Version Control menu functions (for example, Check In, Validate, and Release) in KRM or KTE.
4. Using the Install function on the Version Control menu, load the revised recipe file and all required supporting files into a staging directory for each networked system. The files are integrated into the production directory of each test system at the start of the next lot to be tested.
5. Run the revised recipe from the Keithley Test Execution Engine (KTXE). KTXE adds an execution log to the output file. The execution log records all files and file versions that were used when executing the recipe, resulting in a traceable record.

Knowing the file versions used in a particular production run allows you to recreate a recipe (called a specific recipe) and rerun identical tests with identical test conditions for troubleshooting.

For detailed information about version control, see the Version control (on page 8-1) section in this manual.

Recipe management and version control terminology

The following terms are used in this documentation in the context of Keithley Recipe Manager (KRM) and version control.

Term	Description
Archive	Also called the revision control directory. A directory that securely stores every version of every test recipe and supporting files that are in use and have been used in the past. There is only one archive in a Keithley Recipe Manager (KRM) version-control system; all test-system files are channeled through it to production and to and from development.
Automation interface	The integration of recipes using the SECS/GEM automation option.
Bundle	In the context of the KRM, a group of files that is collected in a single `.tar` file so that certain actions, such as file transfers, can be accomplished more easily. See also .tar.
Check in	An engineering mode operation in Keithley Recipe Manager (KRM) or a Keithley Test Environment (KTE) tool that places a recipe or supporting file in the archive. Use this operation after checking out and modifying a recipe or file. When you check in a file or recipe, it is assigned a new revision number. The lock for a checked-in file is removed so that other users can check out the file. See also Engineering mode, Check out, and Lock.
Check out	An engineering mode operation that places a copy of an archive file into the local area, locks the file in the archive, and makes the file writable. The lock prevents the check out and modification of the file by another person. See also Engineering mode and Check in.
CopyOfProd	A directory that stores a complete, unbundled set of the presently installed PROD-labeled recipes and supporting files, including all updates contained in the staging directory file bundles. See also PROD, Staging area, and Bundle.
Compare	An engineering mode operation in KRM or a KTE tool that determines the differences between versions of a file. See also diffutils.
Deliver	A combined operation on the Version Control menu that loads a PROD-labeled recipe and validates it, checks for locks, checks-in the file and releases it, and installs it in the transfer staging area.
diffutils	Difference utilities. Software used by KRM version control to compare versions of recipes and supporting files, identify the differences, and record the differences in a special file.
Engineering mode	KRM development mode, which is configured for use by engineering or other authorized technical personnel.
Fetch	To check out a read-only version of a file without locking the file in the archive. See also Check out, Lock.
Head	The most recent version of a recipe or supporting file. The head version is normally the version used for production.
Install	A KRM operation that bundles new and updated files from the archive (for recipes, includes all necessary supporting files) and places these bundles (.tar files) in staging directories. Test systems automatically transfer installed files when ready and use them for production. See also Bundle, Supporting file, staging directory.
Install record file	See InstallRecord.log.
InstallRecord.log	A file in the archive that contains a history of each file installation operation for each file bundle in each staging directory.
Install request file	See InstallRequest.log.

Term	Description
InstallRequest.log	A file in the archive that contains a list of files that have been released but not yet installed on test systems. Entries in the InstallRequest.log file can include both recipe files and supporting files.
krm.ini	A user-editable file that configures the fields of the KRM Recipe Contents tab and allows you to define an alternate screen term for the most recent version of a file (instead of the head version).
krm_testers.ini	A file that lists all of the test systems on which KRM installs recipes and supporting files.
KRM	Keithley Recipe Manager. See Recipe Manager.
Label	A unique mark in a file (for example, a recipe or project file) that allows it to be associated with other files.
<LibraryName>_settings.ini	A file that stores the visible and hidden status and library dependencies information for a user library.
Library prototypes file	A file that contains the C-language function prototypes for all modules in a user library. The library prototypes file is regenerated every time the library is built.
License manager	Prevents software from being run without the presence of a coded key, which is purchased as part of a license agreement.
Load recipe	A KRM engineering mode operation that makes read-only copies of the latest-version supporting files for a specified recipe in the archive and places them in the presently active local area. No locks are placed on the files in the archive. See also Supporting file, Local area, and Lock.
Local areas	A local area is made up of a series of directory locations that are used to hold recipe files and supporting files during development.
Lock	A label that is set for a recipe or supporting file in the archive that prevents check out and modification of the file by another user. See also Check out.
Normal production area	A test system directory that stores unbundled normal (released for production) recipes and all needed supporting files. See also Normal recipe, Bundle.
Normal recipe	A recipe that uses PROD-labeled (released for production) supporting files exclusively. See also PROD, Specific recipe.
Operator mode	The production mode of KRM configured specifically for use by an operator.
Process	A defined grouping of products having a common set of test requirements.
PROD	A label that is applied to a file in the archive that is released for production. See also Label.
Product family	A unique identifier for one of the products that is grouped under a particular process. See also Process.
Project	A predetermined directory structure in the local area for storing files for multiple recipes, as defined by a collection of environment variables. See also Local area.
RCS	The revision control system used by KRM.
Recipe	A recipe defines all the information you need to run a particular test. The filename for a recipe has a `.krf` (Keithley recipe file) extension and includes the name of the process and product family under which it is grouped.
Recipe Manager	Keithley Recipe Manager (KRM) is the primary tool for combined recipe version control and file development. The user interface allows you to select recipes, product families, processes, and primary supporting files.

Term	Description
Recipe navigator	The left pane in the KRM interface. It displays all processes, product families, and recipes that are stored in the local area, archive, CopyOfProd directory, and production areas of a test system. Visibility of files depends on the KRM operational mode and view option.
Reference product	An initial set of files containing the basic test requirements for all products within a process.
Reference recipe	A recipe containing elements that meet a variety of common test requirements for a particular product family. The recipe provides a starting point when developing new recipes within the family. A reference recipe is typically available for every product family that is displayed in the KRM recipe navigator pane.
Release	A KRM operation that assigns a PROD label to a production-ready recipe or supporting file. See also Remove label, PROD.
Remove label	A KRM operation that cancels the release operation for a recipe or supporting file by removing its PROD label. See also Release and PROD.
Revert	A KRM operation that discards the changes made to a checked-out file and returns the status of the file to the most recent (head) version. See also Check out, Head.
Revision control directory	See Archive.
Specific production area	A test system directory that stores the installed specific (non-PROD labeled) recipes. Each recipe is bundled with all of its supporting files. See also Specific recipe, Bundle.
Specific recipe	A recipe that uses at least one non-PROD labeled supporting file. See also PROD, Normal recipe.
Staging directory	A directory that stores installed recipes and other files for a test system until the test system is ready to retrieve them and update itself. A tester is updated when it has transferred all files from its staging directory to its normal and specific production areas.
Supporting file	A KTE file (for example, `.cpf`, `.klf`, `.wdf`, or `.ktm`) or a user library file that is used by a recipe.
.tar	The extension for a tape archive file, a type of Linux® file that combines multiple files into one (a file bundle). A `.tar` file may be untarred (opened) to separate it into its component files. See also Bundle.
Transfer and backup staging directory	A staging directory that is unassociated with a test system that contains bundles of the same files that are used by every updated tester. This directory provides a backup of current production files and allows transfer of recipe bundles to new test systems and client sites.
Untar	To separate a `.tar` file into its component files.
Validate recipe	An engineering mode operation of Keithley Recipe Manager (KRM) that checks the usability of a recipe, without starting hardware execution.
Version control	KRM features that control storage, retrieval, modification, and use of every recipe and supporting file version that is being used in the present and has been used in the past. Version control provides traceability for the algorithms, input conditions, and collected data for a test each time that it is executed.

Section 8

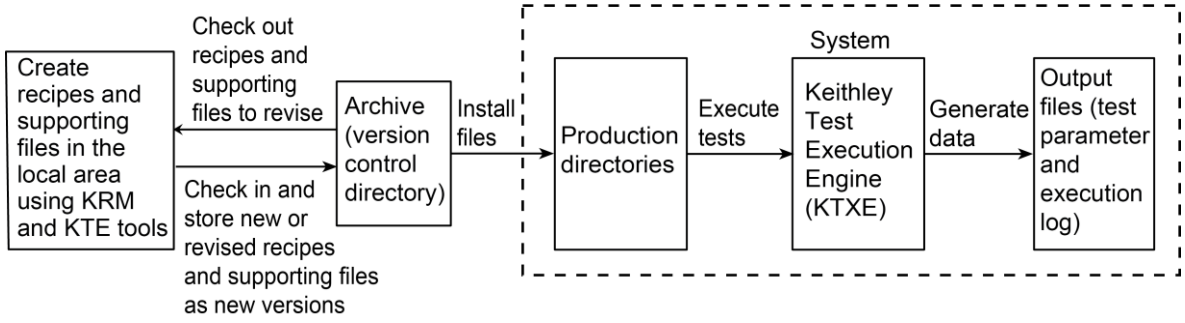
Version control

In this section:

Overview

Version control on the Keithley Parametric Test Systems tracks changes to Keithley Test Environment (KTE) files and Keithley Recipe Manager (KRM) files. The following figure shows the revision path of recipe files and supporting files through the KRM version control system, from engineering through production.

Figure 187: The version control process



The following list describes the key elements in the revision path:

1. Files, revisions, and other information for all versions of recipes and supporting files (including `.cpf`, `.wdf`, `.klf`, and user libraries) are stored in the archive. The archive is a protected directory that is located on the system server.
2. A file that is to be revised is checked out of the archive into a local area. The local area is a directory structure reserved for recipe development.
3. A checked-out recipe is revised in KRM. A checked-out supporting file is revised using the appropriate KTE tool, such as the Wafer Description Utility (WDU) or Keithley Test Plan Manager (KTPM).
4. Version Control menu functions in the KRM or KTE tool, such as Check In, Validate, and Release, are used to update the archive.
5. Using the Version Control menu Install function results in the revised file, and all required supporting files are integrated into the production directory of a client system at the start of the next lot to be tested.[7] This integration occurs for each installed client system.
6. The Keithley Test Execution Engine (KTXE) runs the revised recipe or runs a recipe that uses the revised supporting file. KTXE adds an execution log to the output file. The execution log records all files and file versions that were used when executing the recipe, resulting in a traceable record.

NOTE

Keeping a record of the file version used in production will allow the recipe to be recreated later as a specific recipe. A specific recipe can be created to rerun identical tests with identical test conditions and algorithms because the original file versions are known and available in the archive. For more information, see Understanding normal and specific recipes (on page 8-10).

[7] These files are temporarily stored in staging areas, one for each tester.

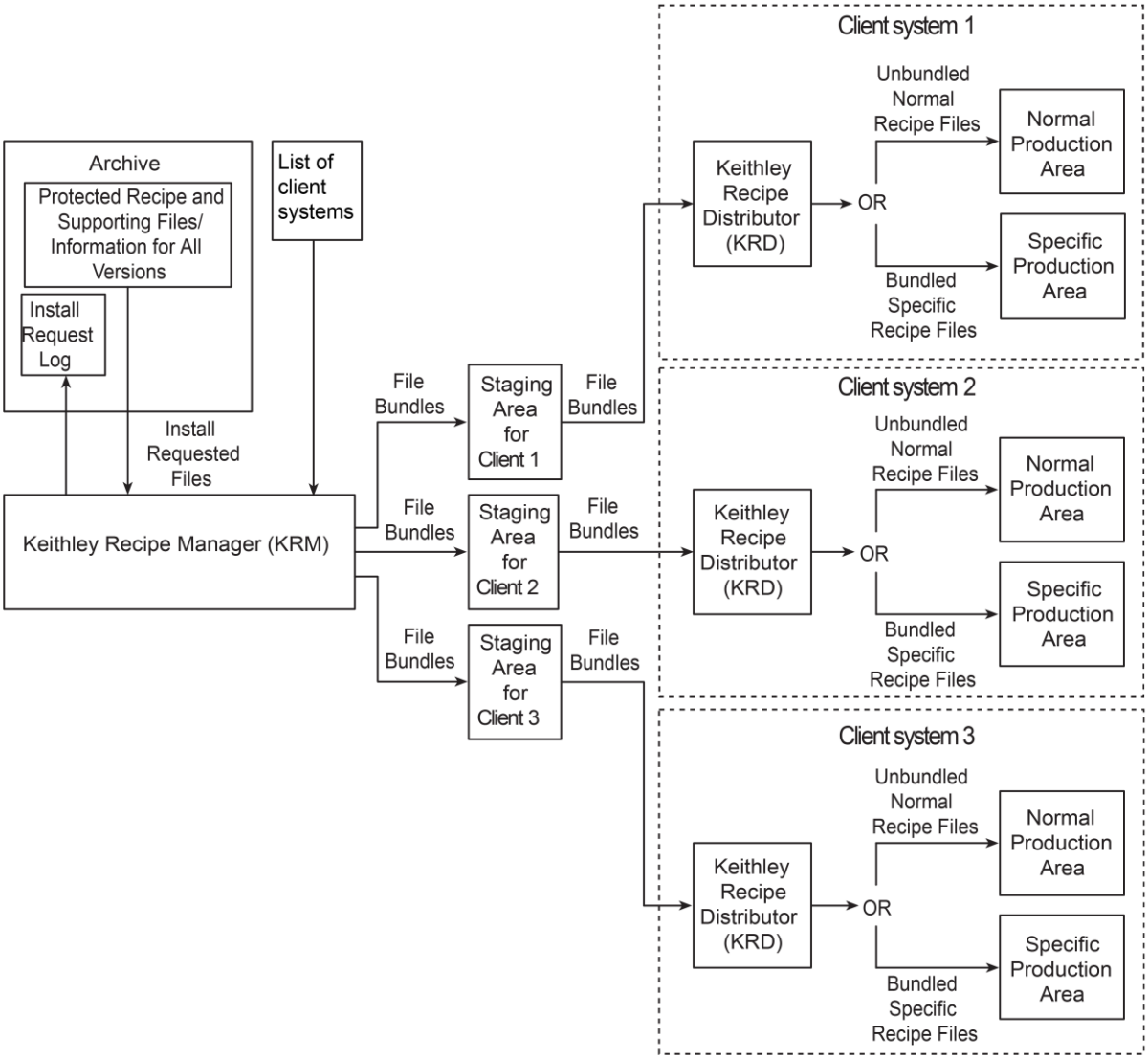
Version control in the production environment

The following topics describe version control in the production environment.

Revised file movement the production environment

The following figure and list describe the movement of revised files into the production environment.

Figure 188: Version-controlled movement of revised files into the production environment



1. An `InstallRequest.log` file maintains a list of all files that are waiting to be installed from the archive.
2. A Keithley Recipe Manager (KRM) Install action creates, from the archive, bundles for the waiting recipes and individual files and places the bundles into staging directories - one for each client system. Note that:
   - KRM determines the client systems to be updated using a list of clients, which is a file that is created as part of the system software distribution and is maintained within KRM.
   - The staging directories are located in the server, along with the archive. This ensures that KRM can update all staging directories when requested, regardless of client system activity.

NOTE

A revised recipe is bundled with all of its supporting files. Revised individual Keithley Test Environment (KTE) files may be bundled separately.

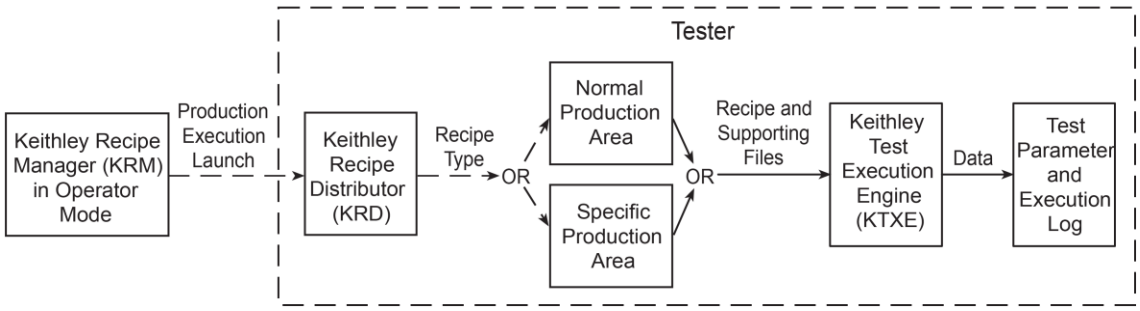
3. Each client system, when it is ready, retrieves the bundle from its staging directory. The transfer occurs immediately before the client system starts its next run or after the completion of support or preventive maintenance activities. This update mechanism precludes possible errors that could occur when attempting to update client system files during lot testing.

   The file bundle is transferred into either a normal or specific production area:
   - Normal recipes are stored in unbundled form in the normal production area. Normal recipes call only supporting files that are labeled for production.
   - Specific recipes are stored as bundles in the specific production area. Specific recipes call one or more supporting files that are not labeled for production. Specific files are typically created when a previously run version needs to be rerun but no longer qualifies as the Normal production version.

The recipe execution process in operator mode

The following figure illustrates execution of a recipe in Keithley Recipe Manager (KRM) operator mode. A description of the process follows the figure.

Figure 189: Recipe execution using KRM in Operator mode



Recipe execution in operator mode proceeds as follows:

1. Log in and start KRM in operator mode.
2. In KRM, select a recipe using the recipe navigator, which displays a graphical hierarchy of processes, product families, and recipes.
3. Once the recipe is selected, select **Execute Recipe**, which initiates the following:
   - Loads the selected recipe from the normal or specific production directory.
   - Runs the Keithley Test Execution Engine (KTXE), generating output files.

The above sequence can also be executed using the automation link option. However, normal and specific recipes must be created in KRM, which cannot be done over the automation link.

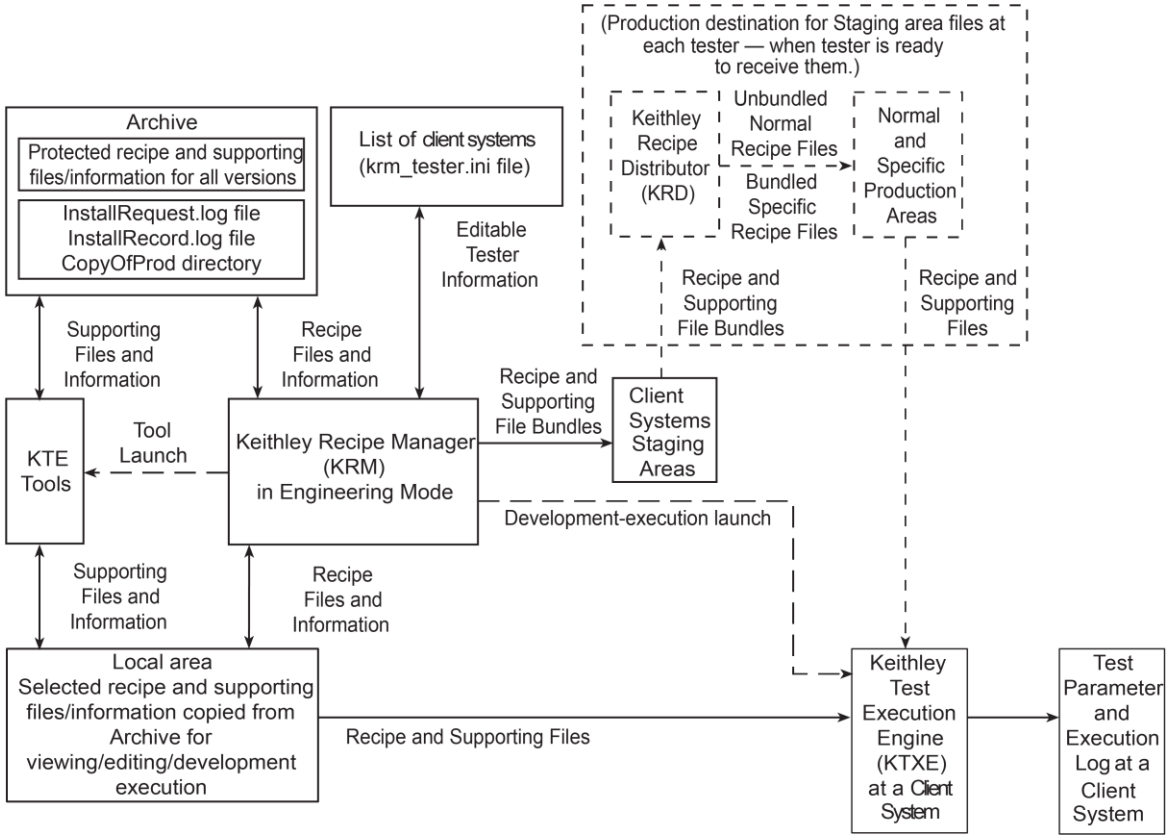
Version control in the engineering environment

A common engineering mode procedure is to modify an existing recipe and its supporting files, test the changes, and release the revised files into production. This commonly requires the following:

- Accessing the archive to check out appropriate files.
- Changing and testing the files until the required functionality is attained.
- Returning the updated files to the archive for subsequent release and installation into production.

The following figure and procedure summarize the use of version control in Keithley Recipe Manager (KRM) for engineering/development requirements.

Figure 190: KRM Version control engineering development functionality



The following steps show the process for the revision and version control of a single recipe or supporting file:

1. Start KRM in engineering mode.
2. Using KRM, select either of the following:
   - A recipe to be revised.
   - A recipe that calls the supporting file to be revised.
3. Using KRM, check out the recipe or supporting file from the archive into a local area. The local area is a directory structure that is reserved for development and is defined by a project.
4. Revise a recipe file directly in KRM. Revise a supporting file using the necessary Keithley Test Environment (KTE) tool, after starting it from KRM (for example, to revise a `.wdf` file, start the Wafer Description Utility (WDU).
5. Check in the revised file to the archive. For a recipe, use the KRM Version Control menu. For a supporting file, use the appropriate KTE tool Version Control menu.

NOTE

You can access the KTE tools through KRM. For more information, see Using Keithley Recipe Manager with KTE tools (on page 7-31).

6. If the revised file is a recipe, validate the recipe from within KRM. You can choose to execute the recipe after validating.

NOTE

KRM is the only tool that can validate an entire recipe when all supporting files are also loaded into the local area.

7. KRM is the only tool that can execute an entire recipe. KRM can start KTXE for the recipe that is selected in KRM's Recipe Navigator. This allows test parameter and execution log output to be verified before installing the recipe into production.
8. Release the file, assigning it a production (PROD) label.
9. Install the revised file with the Install function in the KRM Version Control menu. Running the Install function results in the revised file and any required supporting files are placed into all client system staging directories.

NOTE

KRM is the only tool that can install files that have been previously released but not yet installed.

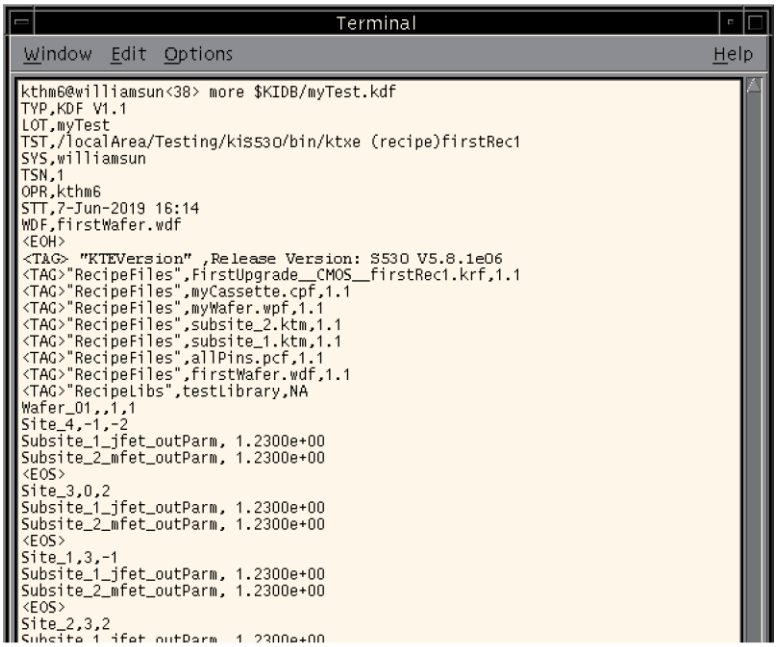
When a client system is ready, the Keithley Recipe Distributor (KRD) moves all installed files from the client system's staging directories to the client system's normal and specific production area. From here, recipes are executed during production.

File version tracking

File version tracking is a key feature of the Keithley Recipe Manager (KRM). Checking in an updated copy of a file creates a new version in the archive. KRM increments the version number for the file and, upon release, assigns it a production-ready (PROD) label. The version control system tracks all versions using the tools and files that are listed below.

- The Recipe Manager (KRM), `.krf` files, and `krm.ini`
- The Revision Control System (RCS) archive
- `$KI_ARCHIVE/InstallReq.log`
- `$KIHOME/krm_testers.ini`
- Client system staging directories
- Normal production area
- Specific production area
- Version Control menu in all Keithley Test Environment (KTE) data management tools (for example, to KITT, KULT, or WDU)
- Updates the version information for existing KTE files.
- Adds a production execution log to the data file (`.kdf`). Each time that a recipe is run, the log saves a record of the files and file versions that are used to run the recipe. The following figure highlights the execution-log part of an example `.kdf` file.

Figure 191: Production execution log part of a .kdf file



The primary version control areas

Before developing recipes, you should understand the primary areas of a Keithley Recipe Manager (KRM) version control system. The following topics describe these areas.

Local areas

A local area is made up of a series of directory locations that are used to hold recipe files and supporting files during development.

NOTE

Periodically, you should delete files a local area to prevent the accumulation of files that are irrelevant to current development. For information about deleting files, refer to Deleting items from the local area (on page 7-64).

The Keithley Recipe Manager (KRM) version-control system needs one local area, a default version of which is created when KTE is installed. However, several local areas may be created and used to meet the needs of multiple developers. Each developer can have one or more local areas.

A local area is created when you create a project using the `make_project` command-line utility. A local area is selected either using the file selection buttons in the KRM interface or by using the `select_project` command-line utility. For more information about projects, refer to Selecting a project (on page 7-29) and Setting up a project (on page 7-24).

Archive

The archive, sometimes called the revision control directory, is a directory that contains the following:

- Master copies of all of the recipe and supporting files in the Keithley Recipe Manager version-control system
- Version-numbered change files that define all revisions to all recipe and supporting files
- `InstallRequest.log` and `InstallRecord.log` files

The archive is a permanent repository for every file. All recipe and file development and production testing is done using copies of files from the archive. If a changed file is checked in to the archive, the changes to the file are stored in the archive with a new revision number.

There is only one archive in the system, which may be used by any number of client systems.

Understanding the head version and the PROD label of a file

- **Head:** Keithley Recipe Manager (KRM) normally designates the latest version of a recipe or supporting file as head. The term head is used unless, in the `krm.ini` (on page 7-89) file, you specify a different designation for the latest version. For information about the krm.ini file, see Configuring user-defined recipe contents fields and terminology (on page 7-89).
- **PROD:** A recipe must always be labeled PROD, using a release action, before it can be used for production testing.

A supporting file must also be labeled PROD, using a release action, before it can be used in a normal recipe for normal production testing. However, a supporting file does not need to be labeled PROD if it is used for a specific recipe.

Commonly, the head version of a recipe or supporting file is also released as the PROD labeled version. However, release of an earlier version as the PROD version is allowed.

Understanding normal and specific recipes

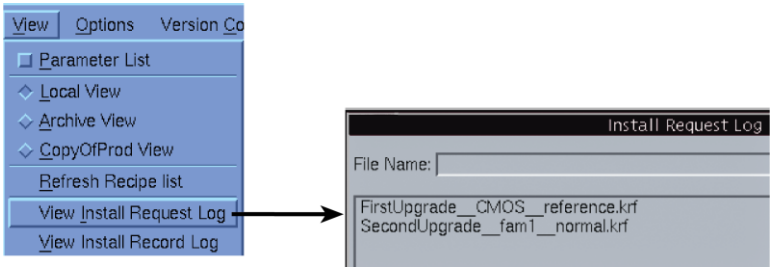
- **Normal:** A normal recipe calls only PROD labeled versions of supporting files.
- **Specific:** A specific recipe calls at least one non-PROD labeled version of a supporting file, commonly for diagnostic purposes. For example, if, after updating a supporting file, recipe test results are unusual, you can:
  1. Create a specific recipe that calls the previous version of the supporting file.
  2. Rerun the test.
  3. Compare the results.

To create a specific recipe, start with a normal recipe and reset the revision labels of one or more supporting files. For more information, see Creating a specific recipe (on page 7-45). Specific recipes are distributed, stored, and run with special processing. Refer to Client systems (on page 8-27) for information.

Understanding the archive InstallRequest.log file

The `InstallRequest.log` file contains a list of files that have been released. For example, these files could have been released by selecting **Version Control > Release**, but they have not yet been installed on a system. Entries in the `InstallRequest.log` file can include both recipe files and supporting files.

Figure 192: Example of the View Install Request log action



The Install Request file includes all edited files that have been released using **Version Control > Release**, but have not yet been installed on a tester. Entries in the Install Request file can include recipe files (`.krf`) and user libraries and supporting files (`.cpf`, `.klf`, `.wdf`, `.ktm`, and other files).

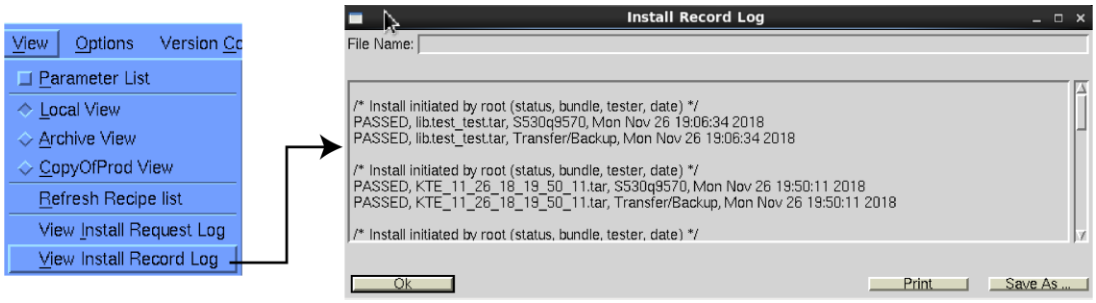
Understanding the archive InstallRecord.log file

The `InstallRecord.log` file contains a history of each install operation, for each file bundle in each staging directory, including the following:

- The installation status (pass or fail) for the bundle
- The file-bundle name
- The staging directory where the file bundle was installed; this is identified by the name of the corresponding system
- The date and time of the installation

You can view the `InstallRecord.log` file using the **View > View Install Record File** menu item.

Figure 193: Viewing the InstallRecord.log file



Understanding the archive CopyOfProd directory

The `CopyOfProd` directory stores a complete, unbundled set of the currently installed PROD-labeled recipes and supporting files, including all updates contained in the staging area file bundles. If a new client system is added to the system, you can copy the entire set of production files to the new client system from the `CopyOfProd` directory.

NOTE

The `CopyOfProd` files and the files on a given client system are not identical if the client system's staging area bundles have not yet been transferred.

The `CopyOfProd` directory is updated every time that an install operation places file bundles into the staging directories. See Staging directories (on page 7-21) and Releasing and installing a recipe or supporting file (on page 7-59) for more information.

You can review the recipes and primary supporting files that are stored in the `CopyOfProd` directory by selecting **View > CopyOfProd**. This displays a Keithley Recipe Manager (KRM) window that looks like the local view and archive view windows, but displays the recipes and supporting files that are in the `CopyOfProd` directory. This directory contains a complete, unbundled set of the presently installed PROD-labeled recipes and supporting files.

The CopyOfProd view is read-only, and the menu selections are the same as for the archive view.

Staging directories

A staging directory is a directory that temporarily stores installed recipes and supporting files as file bundles (`.tar` files). A recipe bundle stores a complete, production-ready normal or specific recipe and all of its supporting files. Other types of bundles contain separate Keithley Test Environment (KTE) files or user libraries. There is a staging directory for each client system, and all staging directories contain identical bundles.

Staging provides the following benefits:

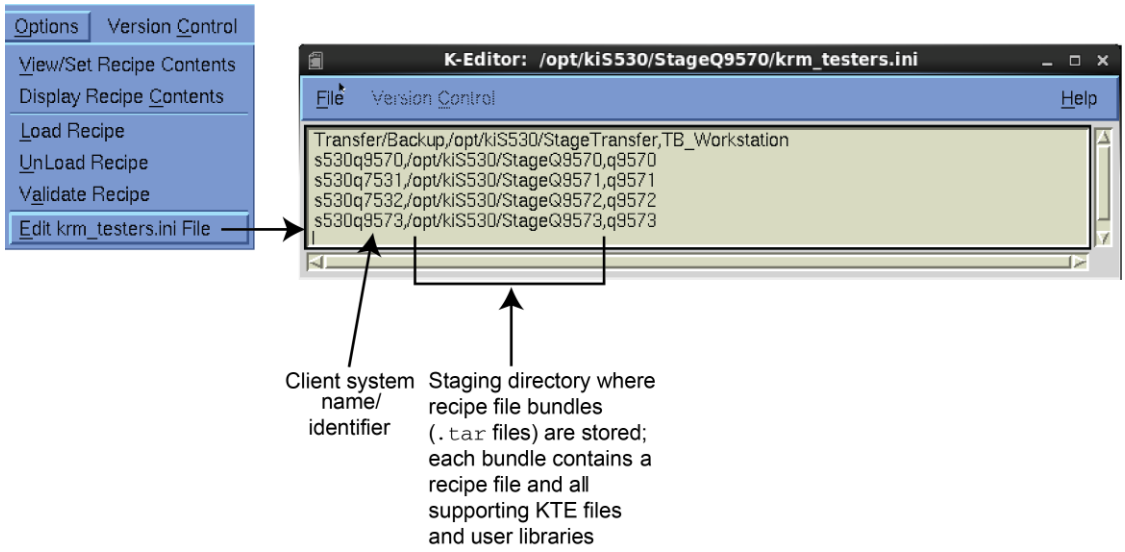
- Allows developers to install new and updated files into temporary locations, at any time, without disrupting in-process client system operations.
- Allows the client systems to retrieve new and updated files when they are ready.

It is helpful to set up at least one extra staging directory that is unassociated with a client system. The extra staging directories can be used to transfer recipe bundles to remote sites or to other areas for archiving or backup. Set up such a staging directory in `krm_testers.ini` by setting up a nonexistent client system (refer to Editing the krm_testers.ini file (on page 8-13)).

Editing the krm_testers.ini file

The `krm_testers.ini` file, located in the directory pointed to by `$KIHOME`, lists the testers that are included in your system. To view and edit the `krm_testers.ini` file, select the Keithley Recipe manager (KRM) **Options > Edit krm_testers.ini** file menu item, which opens the window pictured in the following figure.

Figure 194: Example krm_testers.ini file



Directory file structures

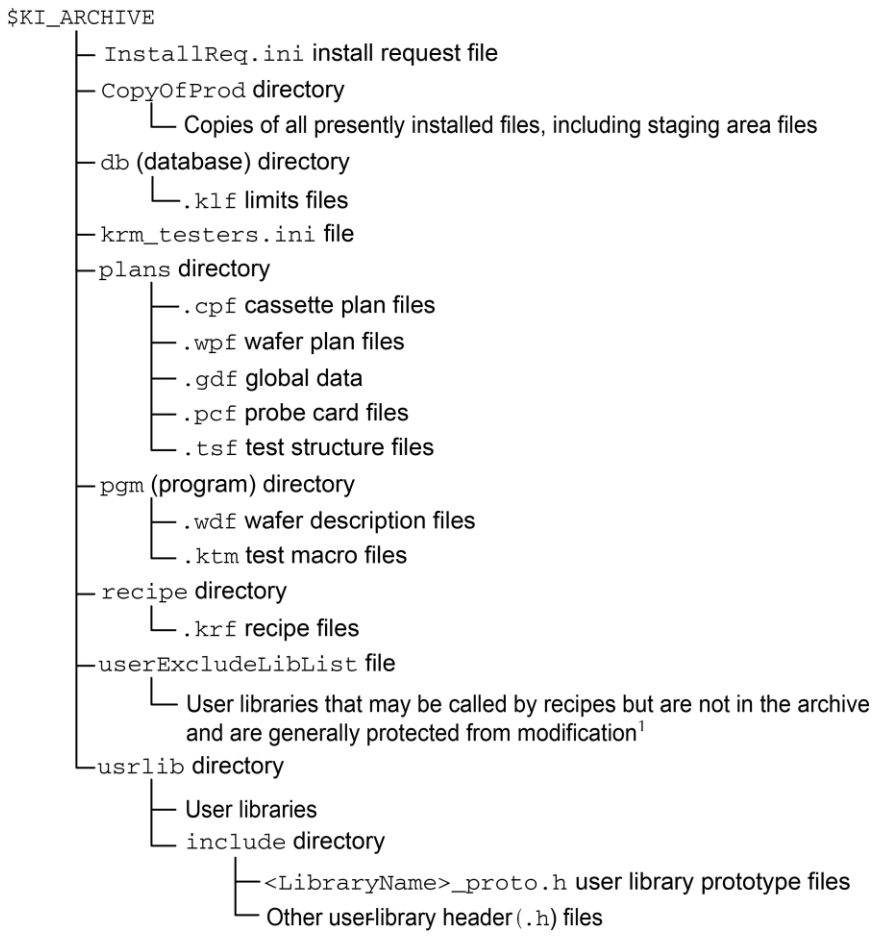
The following topics provide file structure information for the following directories:

- Archive
- Normal and specific production areas in the client system
- Projects

Archive directory structure

The following figure identifies the environment variable for the archive root directory and shows how recipes and supporting files are distributed in this directory.

Figure 195: Archive directory structure



[1] Place user libraries in the ExcludeLibList file using only the library name — exclude the lib prefix and the .so suffix.

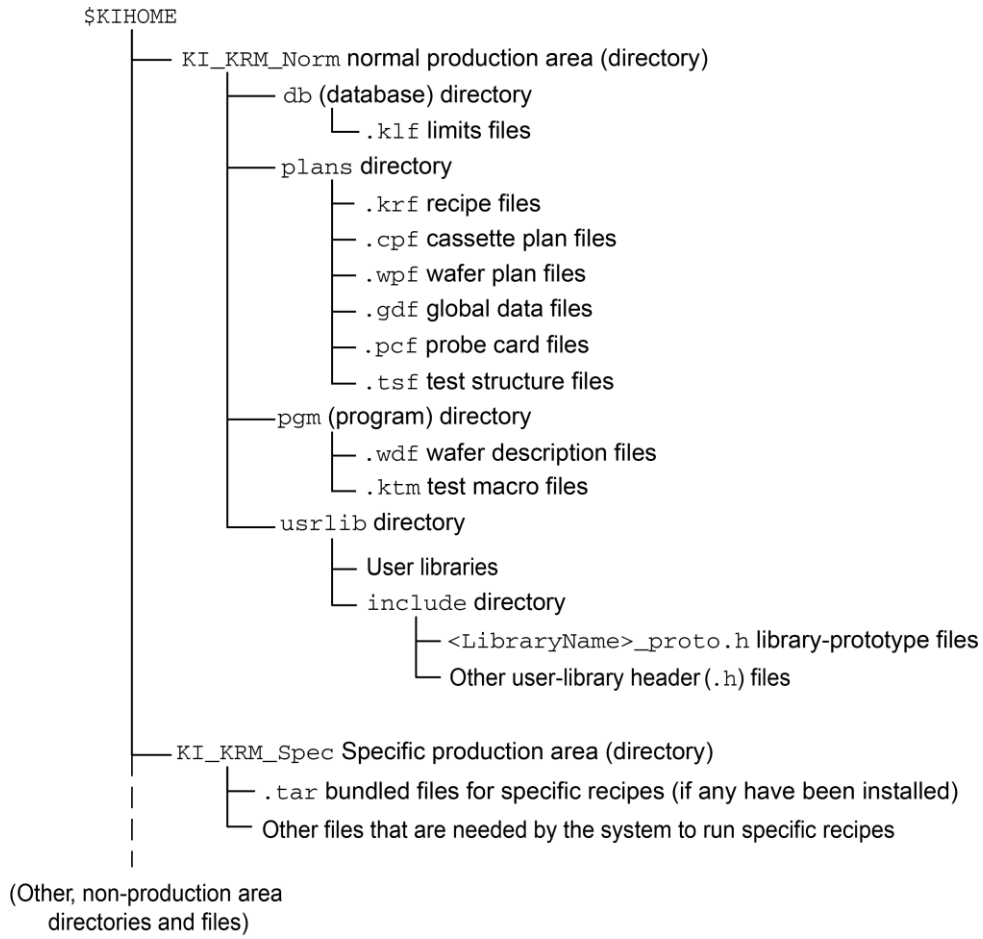
Production directory structure

The following figure identifies the location of the normal and specific production directories and shows how recipes and supporting files are distributed in these directories.

NOTE

In the Normal production area `.krf` files are located in the plans directory, rather than in a separate recipe directory as in the archive.

Figure 196: Production directory structure

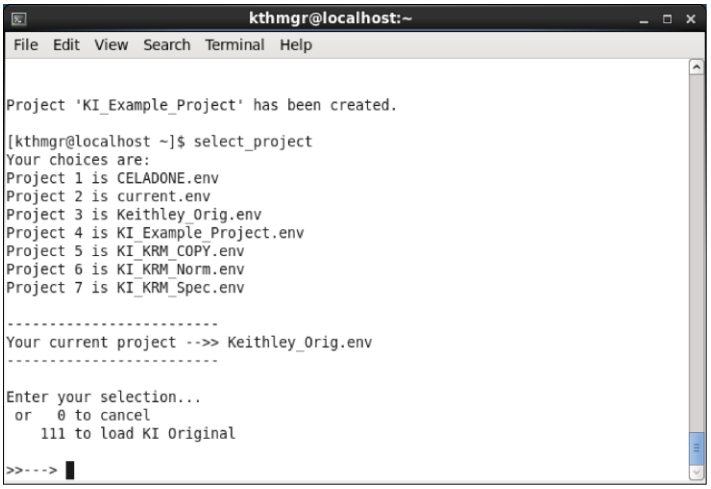


Projects directory structure

A project is a series of environment variables that point to the directory locations of KTE files and user libraries. You can view the names for all available projects that are located in `$KIDAT` as follows:

1. Enter `select_project` on the command line.

Figure 197: Viewing available projects



2. When you have finished viewing, enter `0` to cancel the `select_project` script. Otherwise, you may inadvertently reset the presently active project.

To view the environment variables that comprise the presently active project, along with several other environment variables, enter `ki?` on the command line. For more information, refer to Setting up a project (on page 7-24).

Protecting and monitoring software operation

The following topics describe how to protect and monitor software operation.

Protecting the normal and specific production areas

Reserve the normal and specific production areas for use by the Keithley Recipe Manager (KRM) version-control system. Never use these for development purposes.

Monitoring staging directories for stagnant files

Periodically check the content of each staging directory. A staging directory should always be empty except under the following conditions:

- Recipes or supporting files have recently been installed to the staging directory and the associated client system is busy, and is not ready to transfer the files. A client system is busy when it is occupied with testing or maintenance activities.
- The associated client system is shut down or disabled.

Otherwise, file accumulation in a staging directory indicates that its link to the client system is faulty. A faulty link results in testing with incorrect versions of recipes or supporting files.

Staging directories are typically installed using the `$KIBIN/vc_add_tester_2_server` script, which runs as follows:

- Initially, as part of the Keithley Recipe Manager (KRM) software installation. Refer to Single-system setup (on page 7-8) for more information.
- Subsequently, if you later add clients to the system. Refer to Multiple networked systems setup (on page 7-10) for more information on adding client systems to the system.

If the staging directories were installed using a script, check each staging directory at the location indicated below:

- If you select the default staging directory location for the client system when you execute a script, then the directory location is `$KIHOME/StageQNNNN`, where `NNNN` is the client system's QMO (serial) number.
- If you choose a custom staging directory location for the client system when you execute a script, then the directory location is the entry for this client system. Use the directory you entered on the Multiple networked systems checklist (on page 7-6).

Copying the archive to another site

To create a bundled copy of the archive, and then unbundle and insert the copy at another location:

1. Change the directory to the archive directory by entering `cd $KI_ARCHIVE` at the command prompt.
2. Copy the archive to a `.tar` file by executing the following script:

   `tar cvf <ArchiveFileName>./*`
3. Change the directory to the new site by entering `cd <DestinationDirectory>` at the command prompt (where DestinationDirectory is the path to the new site's directory).
4. At the new site, unbundle and insert the archive copy using the following command:

   `tar xvf <ArchiveFileName.tar>`

Populate the production environment of a new client system

The `$KIBIN/tester_prod_util.pl` Perl script allows you to create a bundle containing the normal and specific production-area files.

You can execute the script on a client system by selecting the **Create Bundle** button. This causes all files in the normal and specific production areas to be bundled together and placed into the client system's `$KI_STAGING_DIR` directory.

You can then move the bundle into a different client system's `$KI_STAGING_DIR` directory and run the `$KIBIN/tester_prod_util.pl` script on that client system. Selecting the bundle name and selecting the **Extract Bundle** button populates the client system's production areas with the contents of the bundle.

Start with a revision offset on initial entry into source control

You can check a file into the archive for the first time with a revision number offset. This is useful when you want to use identical limits files with the same revision number on different model systems.

To set a revision offset on a file on first load to the archive:

1. In the Keithley Test Environment (KTE) tool you are using, select **Version Control > Other Operations > Check In**. The Check In dialog box is displayed.
2. Select the **Set Revision** button.
3. Enter the revision number you want in the dialog box and select **OK**.
4. Select **OK** in the Check In dialog box.

Keithley Recipe Manager (KRM) applies this revision number to the file upon completion of the Check In process.

NOTE

The check-in process fails if the revision number entered is not higher than the current version in the archive.

If you pass the `$KIBIN/checkInLabel` routine the `-r` switch, the supplied argument is used as the initial revision number. You can then use a script to call the `checkInLabel` routine with the appropriate starting revision number for a file. This allows you to populate the archive with files from another facility that have identical revision numbers upon check in.

Transferring production file bundles to the client systems

Files are transferred from staging areas to testers automatically. Each client system automatically updates its normal production area by transferring and unbundling any normal file bundles found in its staging area.

The client system also updates its specific production area by transferring, but not unbundling, any specific recipes that it finds in its staging area.

These automatic updates occur immediately before the client system starts its next run.

Before each run, a tester checks to see if file bundles are present in its Staging area. If it finds bundles, it automatically transfers them, leaving the Staging area empty. The run does not start until all bundles are transferred. However, a tester does *not* transfer bundles during process testing or maintenance activities. Therefore, all testers do not update simultaneously.

When a tester transfers *recipe* bundles, a Keithley Recipe Distributor (KRD) distributes Normal and Specific recipes to separate Normal and Specific production areas. It unbundles Normal recipe bundles into separate files. However, it leaves Specific recipes as bundles. Then, when a tester runs a Specific recipe, it 1) unbundles a copy, 2) executes the copy, and 3) discards the copy. These measures guarantee that, at the tester, each Specific-recipe support file is isolated, both from Normal-recipe supporting files and from all other Specific-recipe supporting files.

Clean up archive files and normal production environment

You can delete archive files and clean up the KI_KRM_Norm production environment. The `$KIBIN/archive_util.pl Perl/Tk` script allows you to do the following:

- Scan the archive for files that do not have the PROD labels. You can then select the file to move to a separate area (`$KI_ARCHIVE/backup directory`) to archive the archive. The moved files will no longer be in the archive, but will still be available using the command-line, for future reference or analysis.
- Scan the `KI_KRM_Norm` production area. Determine for each file whether an archive exists and whether the archive file has a PROD label. If not, the script allows you to delete the file from the production area and the `$KI_ARCHIVE/CopyOfProd` area.

Viewing the history of a file

You can view the revision history, locked or unlocked status, and which versions of files have the head designation and PROD label using most Keithley Test Environment (KTE) tools.

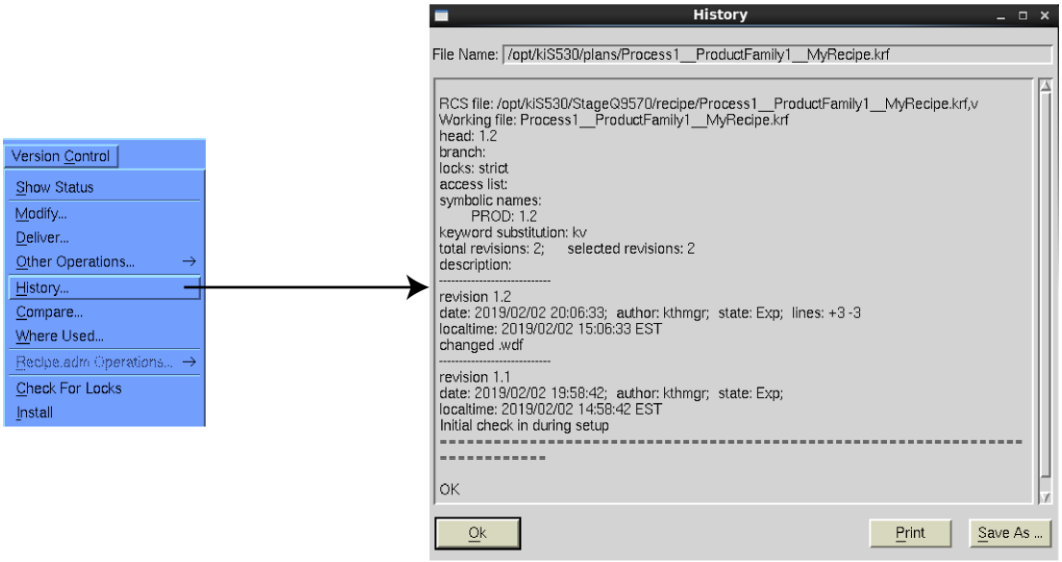
NOTE

You can view the history of a file in Keithley Test Plan Manager (KTPM), Wafer Description Utility (WDU), Keithley Interactive Test Tool (KITT), Limits File Editor (LFE), Parameter Set Editor (PSE), Test Structure File Editor (TSE), Keithley Data Editor (KDE), and Keithley Recipe Manager (KRM) tools.

To view the file history:

1. In the KTE tool you are using, select **Version Control > History**. The History window is displayed.

Figure 198: Viewing the history of a file



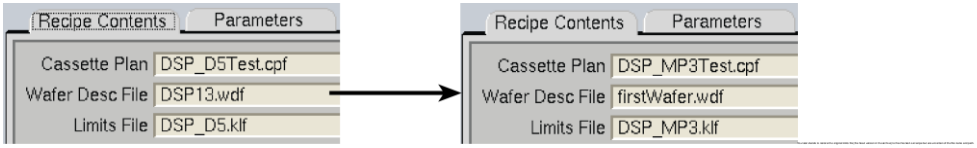
2. After viewing the file history, you can print or save the history by selecting **Print** or **Save As**.
3. Select **OK** to close the History window.

Comparing versions of a file

The compare operation compares one version of a recipe or supporting file with another version of the same file and reports the differences. For example:

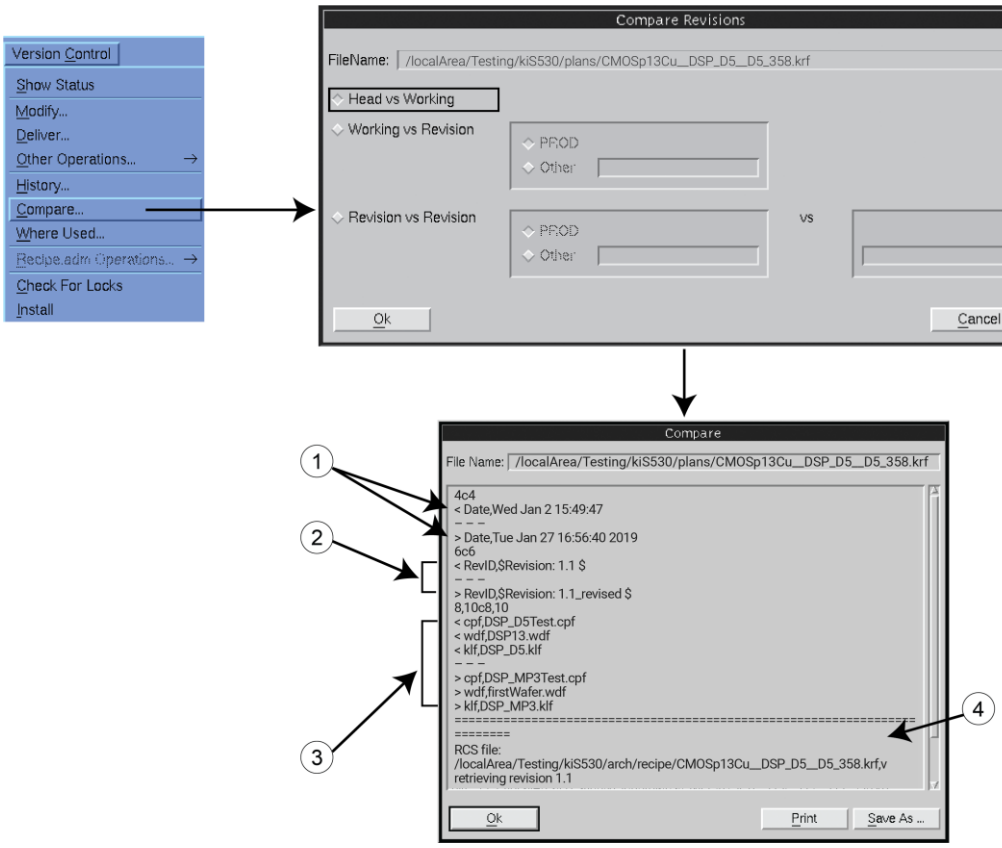
- You check out a recipe from the archive and then change the cassette plan (`.cpf`), the limits file (`.klf`), and the wafer description file (`.wdf`), as shown in the following figure.

Figure 199: Example of saved recipe change in the local area



- You use the **Version Control > Compare** operation to quickly identify the name of the `.wdf` file that was previously used in the recipe. The following figure illustrates the process and shows you the kind of information you can see when you compare files. The table following the figure describes the items in the Compare window.

Figure 200: The compare process



1	Creation dates for the compared versions
2	Identities of the versions being compared; a < symbol signifies an item in the earlier version that was changed in the later version; a > signifies a new or changed item in the later version
3	The list of items that have changed between versions
4	Compare process information (only this displays if no changes were made to the local version)

Viewing where a supporting file is used

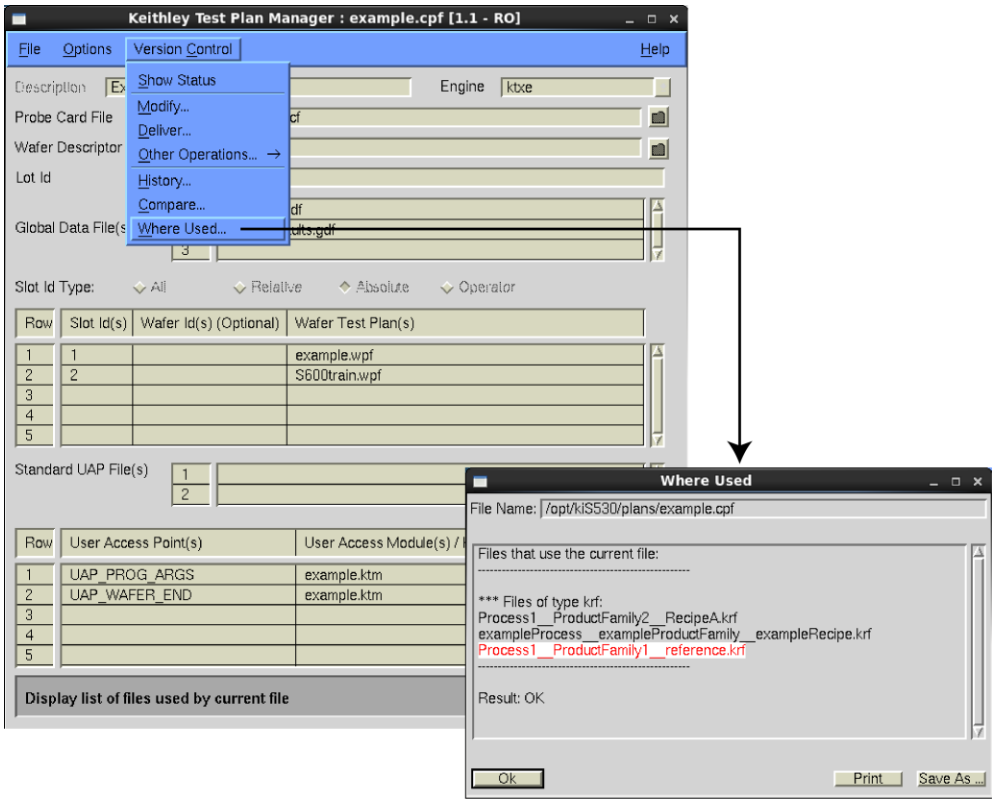
In most Keithley Test Environment (KTE) tools, you can view a list of recipe files that use the presently selected file by selecting **Version Control > Where used**.

The following figure shows the recipes that use the `example.cpf` cassette plan file, which is open in Keithley Test Plan Manager (KTPM).

NOTE

This function is not useful in Keithley Recipe Manager (KRM). If you use it there, the returned list will show `None`.

Figure 201: Using Where Used



Removing a KULT module from production

Removal of a KULT module from the normal production area is a multi-step process. This process is summarized as follows:

1. Fetch the module from the Other Operations menu under Version Control.
2. Load the module from the Options menu.
3. Remove the PROD label from the module.
4. Delete the module from the local KULT library (not from the archive).
5. Check out the Library Prototypes file for modification.
6. Build the local KULT library.
7. Deliver (check in and release) the updated Library Prototypes file.
8. Install the updated KULT library.

This procedure is detailed in the following paragraphs.

Step A: Fetch and load the library to be modified

1. Start the Keithley User Library Tool (KULT).
2. In KULT, select **File > Delete Library**. The Delete Library window opens.
3. Make sure that you do not have a local copy of the library to be modified. If you do, select the library from the list and select **OK**. Otherwise, select **Cancel**.
4. In KULT, select **Version Control > Fetch & Load Library**. The Select Library From Archives window opens.
5. Select the library to be modified.
6. Select **OK**. The Fetch Revision window opens.
7. In the Fetch Revision window, select the **PROD** button.

NOTE

Make sure that you select the **PROD** button (not the default Head selection).

8. Select **OK**. Wait until KULT compiles all modules and builds a local copy of the library.

Step B: Remove the PROD label from the module

1. In the Keithley User Library Tool (KULT), select **File > Open Module**. The module selection window opens.
2. In the module selection window, select the module to be removed from production.
3. Select **OK.** KULT opens the module.
4. In KULT, select **Version Control > Library Module > Other Operations > Remove Label**. The Remove Label window opens.
5. In the Remove Label window, enter the module name, path, and select the **PROD** button.
6. Select **OK**. The Enter Password window appears.
7. In the Enter Password window, enter the same password that you use to release files in the Recipe Manager (the default password is `kthadm`).
8. Select **OK**. This removes the PROD label from the module in the archive.

Step C: Delete the module from the local KULT library

1. In the Keithley User Library Tool (KULT), select **File > Delete Module**. The module selection window opens.
2. In the module selection window, select the module to be deleted.
3. Select **OK**. A confirmation window opens with the message `Are you sure you want to delete module <module_name>.c?`
4. Select **OK**. A message window opens with the message `Module <module_name>.c is read-only. Delete anyway?`
5. Select **Yes**. A reminder to rebuild the library appears.
6. Select **OK**.

Step D: Check out the Library Prototypes file for modification

1. In KULT, select **Version Control > Library Prototypes > Modify**.
2. A message window opens with the message `Override local copy of library prototypes?`
3. Select **OK**.

Step E: Build the local KULT library

In the Keithley User Library Tool (KULT), select **Options > Build Library**. The Library Prototypes file is updated when the library is rebuilt.

Step F: Deliver the updated Library Prototypes file

1. In the Keithley User Library Tool (KULT), select **Version Control > Library Prototypes > Deliver**. A window opens for you to enter a comment.
2. Enter a comment for removing the module from production.
3. Select **OK**. The Enter Password window opens.
4. Enter the same password that you use to release files in the Recipe Manager (`kthmgr` is the default password).

NOTE

For more information about Keithley Recipe Manager passwords, see System and software passwords (on page 7-3).

5. Select **OK**.

Step G: Install the updated KULT library

1. Start Keithley Recipe Manager (KRM).
2. In KRM, select **Version Control > Install**. The Enter Password window opens.
3. Enter the password for installing files to production (the default password is `kthadm`).

NOTE

For more information about Keithley Recipe Manager passwords, see System and software passwords (on page 7-3).

4. Select **OK**. A message window opens with the message `Tester(s) found. Continue Install?`
5. Select **Yes**. The test program shows the following message:
   `Install window shows the installation progress. Some messages may appear in your terminal window as the library is built for installation. Please be patient.`
6. A message window opens with the message `All Files were successfully installed!`
7. Select **OK**.

The revised KULT library and any other newly released files in the archive are installed to the staging directories. From there they are automatically transferred to the client systems at the appropriate times.

Client systems

Beyond their traditional KTE purposes, client systems in a KRM/Version Control system are used as follows:

- When it is ready, each client system in the system automatically retrieves the current set of recipe bundles from its staging directory. A Keithley Recipe Distributor (KRD) transfers the normal and specific recipes to separate normal and specific production areas.
- Normal recipes are unbundled after being transferred. Specific recipes are not unbundled until selected for execution.

For more details, refer to Transferring production file bundles to the client systems (on page 8-19).

Version control menu in KULT

The Version Control menu in KULT resembles the Version Control menus in other KTE tools. However, because KULT works with user libraries and user modules, its main Version Control menu items (and its Header Files submenu items) are different. Those items are covered separately in this section.

Automatic version control functions

By default, version control automatically applies a modify operation to a library prototypes file in the following situations:

- Whenever dependencies or visibility are modified for the corresponding library
- Whenever a module for the corresponding library is created, modified, or deleted

Version control also automatically applies a modify operation to a library settings file whenever dependencies or visibility are modified for the corresponding library. A maintained internal state indicates whether version control has automatically applied modify operations to the library prototypes or library settings files. If it has, the Keithley User Library Tool (KULT) automatically prompts you to perform the following when a deliver operation occurs:

- Build the library.
- Deliver the library prototypes file or library settings file.

NOTE

You can set the `KI_KULT_MAN_VC_MODE` environment variable to disable these automatic actions, maintaining classic version control behavior.

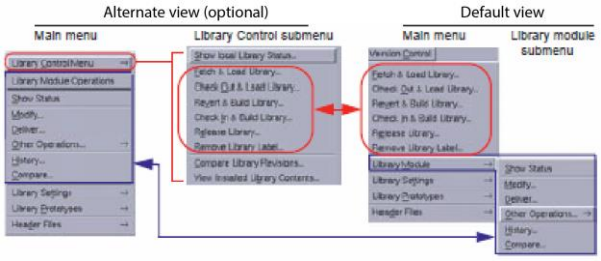
Alternate version-control menu

The Keithley Library Tool (KULT) displays the classic menu layout by default. To enable an alternate layout, set the `KI_KULT_MODULE_MENU` environment variable.

As shown in the following figure, the alternate version-control menu differs from the default version-control menu in the following ways:

- In the default menu view, the top items of the menu are for the user library; those items are now in a submenu.
- In the alternate menu view, the Library Module version control menu is part of the main menu, not in a submenu.

Figure 202: Comparison of default and alternate version-control menu layouts



Library Control submenu in KULT

The Keithley User Library Tool (KULT) includes these added options:

- **Show Local Library Status**: Shows the Version Control status of the local library contents.
- **Compare Library Revisions:** Allows you to compare the contents of two different versions of a `library.adm` file. This information can be used to determine which files have changed between versions of a library. An individual module compare operation can be used to get additional details.
- **View Installed Library Contents:** Allows you to view the contents of the currently installed `library.adm` file.

General KULT version control menu items

The Keithley User Library Tool (KULT) Version Control menu items are:

- Fetch & Load Library
- Check Out & Load Library
- Revert & Build Library
- Check In & Build Library

- Release Library
- Remove Library Label

You should use these menu items to perform version control operations in KULT. They initiate combined operations on entire user library files, rather than on individual files. This saves effort and minimizes the potential for error.

NOTE

The first four menu items conclude with compilation of all user modules in the user library and building of the library. Therefore, some users may prefer to perform separate file-level version control operations in KULT to save time. Refer to Advanced KULT version control menu items (on page 8-34) for more information.

KULT Version Control > Fetch & Load Library

The Fetch & Load Library option allows you to choose a user library from the archive and performs the following:

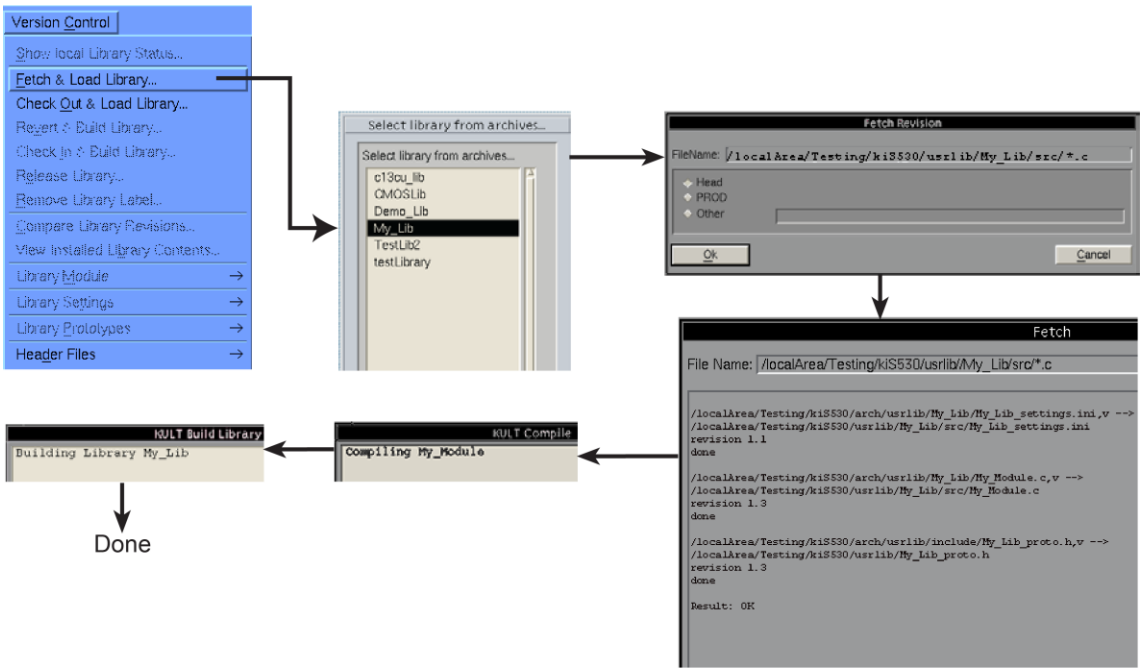
- Places a read-only copy of each file of the selected library into the local area.
- Compiles all library modules and builds the library.

The following figure shows the Fetch & Load Library actions.

NOTE

If the user library exists in your local area, you must first delete it before fetching the archive copy.

Figure 203: Fetch and load library



KULT Version Control > Check Out & Load Library

The Check Out & Load Library item allows you to choose a user library from the archive and performs the following:

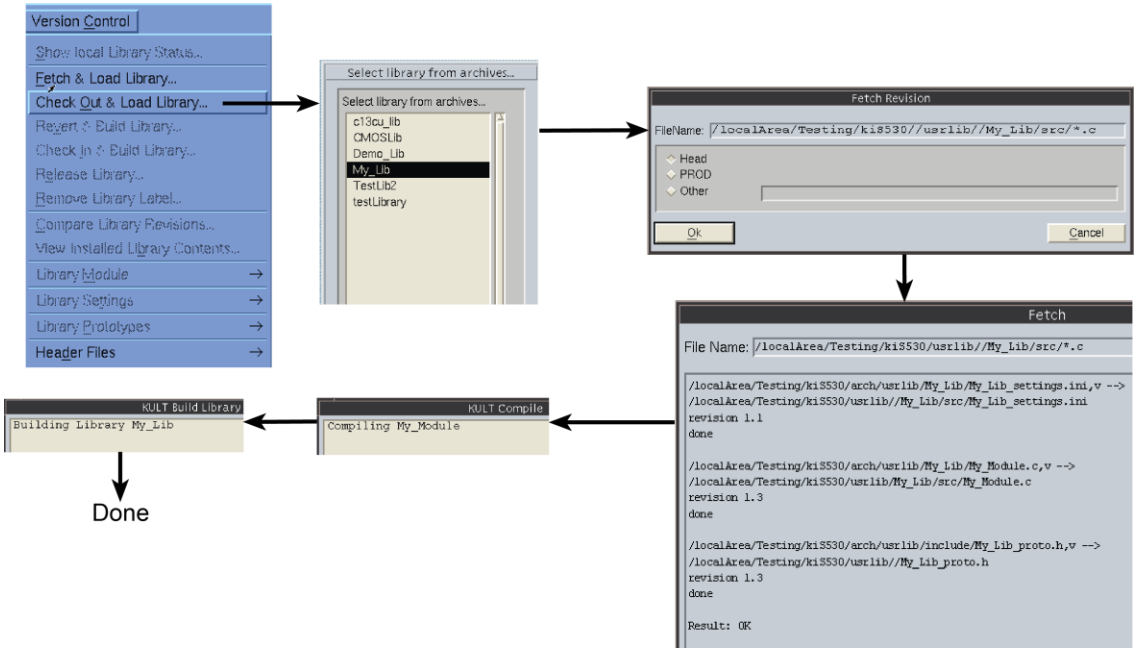
- Locks the archive copy of the library.
- Places a read/write copy of each file of the selected library into the local area.
- Compiles all library modules and builds the library.

The following figure shows the Check Out & Load Library actions.

NOTE

If the user library to be checked out exists in your local area, you must first delete it before checking out the archive copy.

Figure 204: Check Out & Load operation



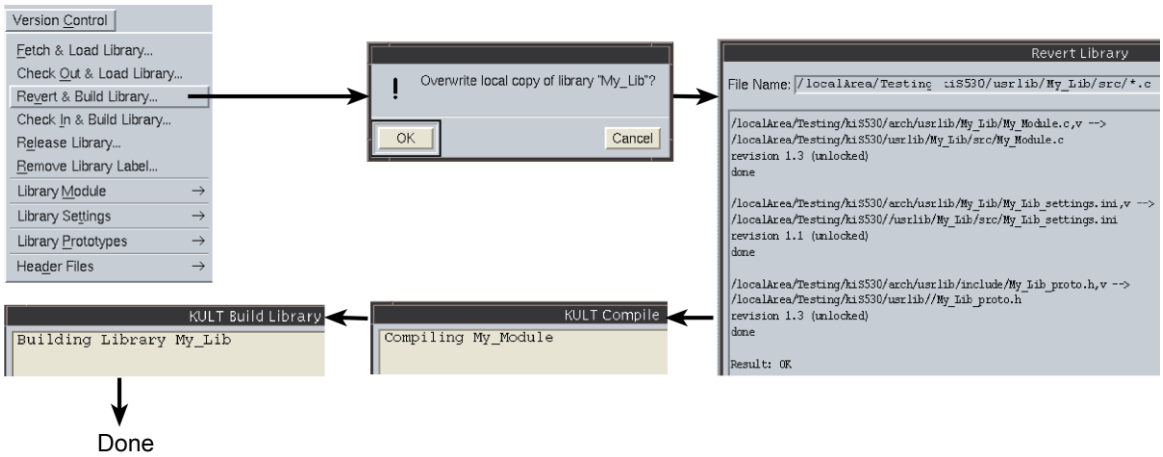
KULT Version Control > Revert & Build Library

The Revert & Build Library item performs the following:

- Restores, to the local area, the Head versions of all library files.
- Sets the local area permissions for the library files back to read-only.
- Unlocks the archive copy of the library files. The Archive is left unchanged.
- Compiles all library modules and builds the library.

The following figure shows the Revert & Build Library actions.

Figure 205: Revert & Build Library in KULT



KULT Version Control > Check In & Build Library

The Check In & Build Library option does the following:

- Places an updated copy of the local area library files into the archive.

NOTE

Modified files trigger an assignment of a new revision number.

- Sets the local permissions for the library files back to read-only.
- Unlocks the library files in the archive.
- Compiles all library modules and builds the library.

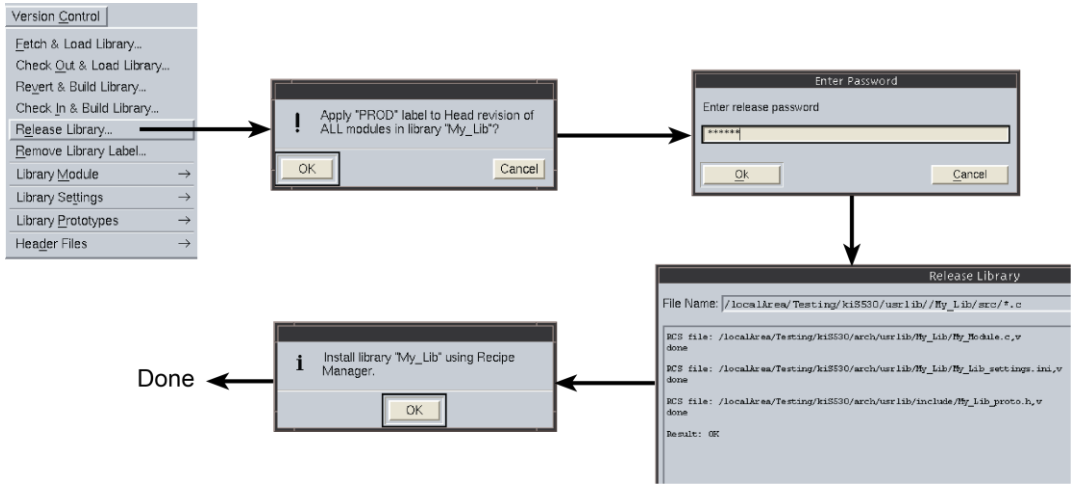
The following figure shows the **Check In & Build Library** on-screen actions.

Figure 206: Check In & Build in KULT

KULT Version Control > Release Library

The Release Library item assigns the PROD label to the Head version of the user library's modules. The following figure shows the Release Library actions.

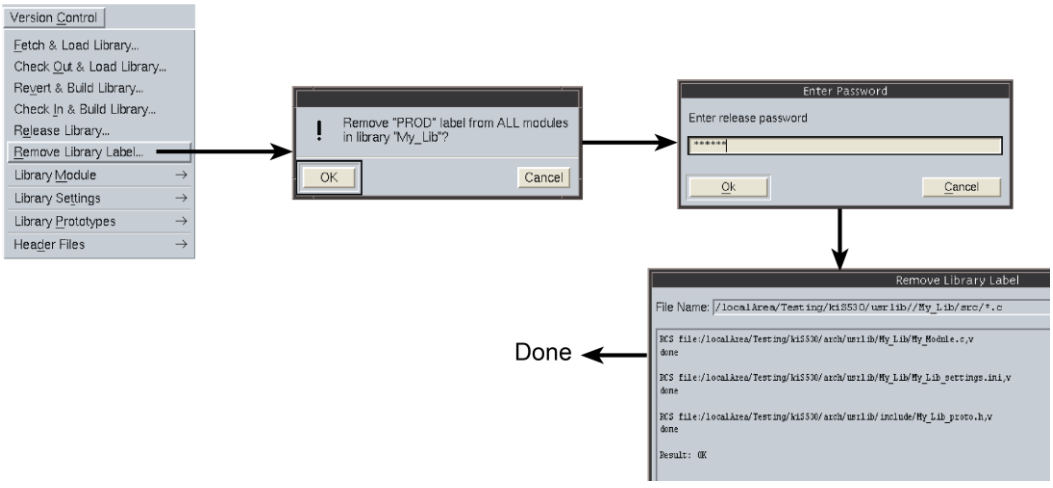
Figure 207: Release Library action in KULT



KULT Version Control > Remove Library Label

The Remove Library Label item reverts a change from the Release Library action. It removes the PROD label from all modules in the user library. The following figure shows the Remove Library actions.

Figure 208: Remove Library Label action in KULT



Advanced KULT version control menu items

The Advanced Version Control menu items are designed for advanced users. Advanced users are experienced KULT users who are knowledgeable and confident in their ability use the files and individual operations involved in KULT version control.

The Advanced KULT Version Control menu items allow you to work individually with user-library modules, as well as with the other types of files that accompany user libraries. The other types of files are as follows:

- Library Settings: There is a `LibrayName_settings.ini` file for each user library. A `LibrayName_settings.ini` file stores these library settings:
  - Visible or Hidden status of the library.
  - Library Dependencies.
- Library Prototypes: A Library Prototypes file contains the C-code function prototypes for all user modules in the user library. The Library Prototypes file is generated or updated every time that the library is built. A copy of this file is installed along with the binary form of the user library in the client systems.
- Header Files: Applies only to custom header files that are created by the user and used in a library.

The following figure shows a Fetch operation that includes a user-library module, a Library Settings file, and Library Prototypes file.

Figure 209: Fetch window showing library settings, module, and library prototypes files

The following four Advanced Version Control menu items allow you to individually check out, change, check in, and release user modules and other user-library files:

- Library Module submenus.
- Library Settings submenus.
- Library Prototypes submenus.
- Header Files submenus.

NOTE

After using the advanced menu operations to modify user-library files, you must manually compile the modules and rebuild the library to make these changes effective.

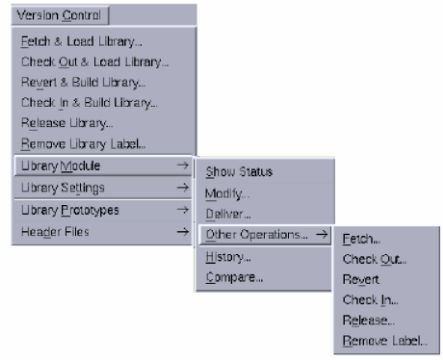
KULT Version Control > Library Module submenus

The Version Control > Library Module submenus are similar to the typical KTE Version Control menus. However, in this case, they enable you to:

- Check out an individual user module
- Edit the module using KULT
- Check in (or revert) the module
- Release the module.

The following figure shows the Version Control > Library Module submenus.

Figure 210: Library module submenus in KULT



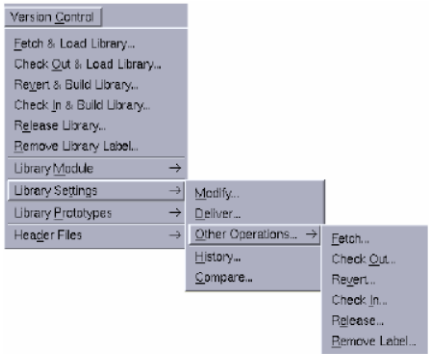
KULT Version Control > Library Settings submenus

The Version Control > Library Settings submenus are similar to the KTE Version Control menus. However, in this case they enable you to:

- Check out the library's settings file
- Change the settings file using the KULT Options menu
- Check in (or revert) the file
- Release the file.

The following figure shows the Version Control > Library Settings submenus.

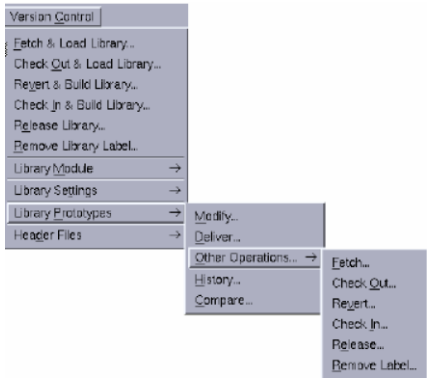
Figure 211: Library Settings submenus in KULT



KULT Version Control > Library Prototypes submenus

The Version Control > Library Prototypes submenus allow you to perform version control operations on the library prototypes file. The following figure shows the Version Control > Library Prototypes submenus.

Figure 212: Library Prototypes submenus in KULT



A library prototypes file contains C-language function prototypes and other information for all modules of the user library. The file is needed for proper operation of the client systems.

You may check out the library prototypes file using either of the following:

- Version Control > Library Prototypes > Modify
- Version Control > Library Prototypes > Other Operations > Check Out

A KULT Build Library operation updates the library prototypes file. If the prototypes file cannot be updated when you request a build, the message shown in the following figure appears.

Figure 213: Library build without prototype file update message

KULT Version Control > Header Files submenu

The KULT Version Control > Header Files > Fetch option allows you to place a read-only copy of a header file from the archive into your local area.

The KULT Version Control > Header Files > Edit item allows you to select a header file in your local area and launch the Keithley Editor. You can directly modify the file and perform typical KTE Version Control operations.

For example, in the Keithley Editor, you could check out the header file and change a value in a `#define` statement. Later, after individually recompiling the modules that use this header file and then rebuilding the library, you could deliver the changes using the Keithley Editor Version Control menu.

Figure 214: Example header file in the edit window

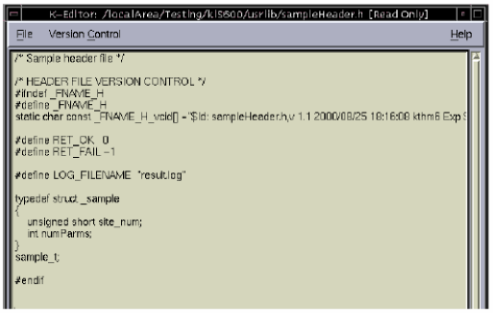


Figure 215: KULT version control header files edit window and menu

Section 9

High-voltage C-V measurements

In this section:

Introduction

This section contains general information about doing basic and advanced high-voltage C-V measurements and provides brief descriptions of the High-Voltage Library (HVLib). For detailed descriptions of the HVLib commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Making high-voltage capacitance-voltage measurements

To bias transistors to high voltage and measure capacitance, any high-voltage capacitance-voltage (C-V) measurement technique must use bias tees to protect instrumentation and devices under test (DUTs) from damage.

Bias tees allow you to mix high-voltage DC bias voltage with an AC signal. A drawback of using bias tees is that it causes degradation of the AC pathway and increased measurement errors. To solve this problem, you can apply compensation factors to measurements to negate the effect of bias tees in high-voltage measurement applications.

The S540 Parametric Test System High-Voltage Library (HVLib) gives you the ability to make basic high-voltage capacitance-voltage measurements with fixed system-level compensation factors and more accurate, advanced high-voltage C-V measurements using preprogrammed system-level compensation factors and user-generated device-level compensation factors.

CMTRs in the S540 system

A capacitance meter (CMTR) is a vector impedance meter that evaluates AC impedance of a device, including both real and imaginary (reactive) parts of the complex impedance.

The AC drive signal is supplied to the device under test (DUT) from the high side of the CMTR, and an automatically balanced bridge on the low side of the CMTR measures the amplitude and phase of the current.

The S540 Power Semiconductor Test System supports several configurations:

- System with no capacitance meters (CMTRs)
- System with one low-voltage CMTR (CMTR1)
- System with one low-voltage CMTR (CMTR1) and one high-voltage CMTR (CMTR2)

For more information about complex AC impedance, see AC impedance (on page 9-4).

Basic and advanced commands

The most commonly used commands are provided in the HVLib in both basic and advanced versions. These commands are described in the following table.

Basic command	Advanced command	Purpose
`hvcv_3term_basic`	`hvcv_3term`	This command measures output capacitance (C_{oss}), input capacitance (C_{iss}), or short-circuit reverse transfer capacitance (C_{rss}) of three-terminal devices.
`hvcv_sweep_basic`	`hvcv_sweep`	This command does a high-voltage capacitance-voltage (C-V) sweep.
`hvcv_test_basic`	`hvcv_test`	This command makes a high-voltage capacitance-voltage (C-V) measurement at a single frequency.

Differences between basic and advanced commands

The main difference between the two versions of these commands is that several parameters are fixed in the basic commands to make it simpler to make high-voltage capacitance-voltage (C-V) measurements. These parameters and their fixed values are:

Parameter	Description	Fixed value
Dut	Device under test; valid options: ■ `dut` = Test the DUT itself with the high-voltage capacitance meter (CMTR) ■ `open` = Characterize the open device using the high-voltage CMTR; this can be done with the chuck down (pins not in contact with the device) ■ `short` = Characterize the short device using the high-voltage CMTR ■ `load` = Characterize the load device using the high-voltage CMTR ■ `shortEx` = Characterize the short device using the low-voltage capacitance meter (CMTR); this data is used to do `CompShort` compensation of the *`loadEx`* device ■ `loadEx` = Measure the load device using the low-voltage CMTR to get the expected value of the `loadEx` device ■ `openEx` = Characterize the open device using the low-voltage CMTR; this data is used to do `CompOpen` compensation of the `loadEx` device	`dut`
Comp_mode	Compensation type: ■ `CompNone` (use this if you do not want to run any compensation or if the *`dut`* parameter is set to anything other than `dut`) ■ `CompOpen` ■ `CompShort` ■ `CompLoad` ■ `CompOpenLoad` ■ `CompShortOpen` ■ `CompShortLoad` ■ `CompShortOpenLoad`	`compNone`
doComp	Specifies whether to do system-level compensation: ■ `0` = Do not do system-level compensation ■ `1` = Do system-level compensation ■ 2 = Do user-created compensation (not available for hvcv_3term and hvcv_3term_basic commands)	`1`

Parameter	Description	Fixed value
doRetest	Specifies whether to remeasure compensation data: ■ 0 = Do not remeasure compensation data ■ 1 = Remeasure compensation data once, then reuse that measurement in any additional calls	1

For a thorough discussion of advanced high-voltage C-V measurement topics, see Advanced high-voltage C-V measurements. For detailed descriptions of each of the HVLib commands, see High-Voltage Library command reference.

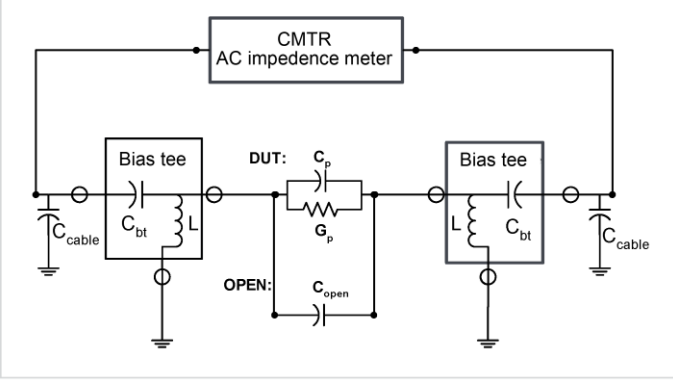
AC impedance

The ratio of the complex AC voltage vector to the current vector provides a complex impedance, which is then converted to specified values using the selected impedance model. The most common impedance models are parallel capacitance and conductance (C_p and G_p) and magnitude and series phase (Z and theta). The High-Voltage Library (HVLib) software automatically makes the calculations to convert the raw CMTR data to the model you choose.

Bias tees and compensation in a two-terminal AC model

Capacitance-voltage (C-V) measurements made with bias tees in the circuit have significant error, and this error must be addressed using compensation factors. For example, if capacitance measurements are made using Keithley Instruments S530-RBT-3KV bias tees with no compensation, measurement error can be as high as three to four percent.

Figure 216: Two-terminal high-voltage capacitance measurement



An impedance analysis shows that the measured value of the impedance (Z_{meas}) can be related to impedance of the device under test (Z_{dut}) using following equation:

$$Z_{meas} = k \times Z_{dut} \,||\, Z_{open} + Z_{short}$$

Where:

- Z_{meas} = Measured impedance
- k = Gain error, $1 + (C_{cable}/C_{bt})^2$
- Z_{dut} = Actual device impedance
- Z_{open} = Measured open parasitic impedance
- Z_{short} = Measured impedance of short device, $2 \times (1/jw \times C_{bt}) \times (1+ C_{cable}/C_{bt})$

You can use this equation to build a compensation model that calculates device capacitance and removes the effects of parasitic capacitances and bias tees. This calculation requires values for the open device (Z_{open}) and short device (Z_{short}) and the value of the gain compensation (k).

Using this model, the effect of the bias tees in the circuit is the same as the gain error. Compensation for the gain error requires measurement of a load standard. In this context, a load standard is a device under test (DUT) of approximately the same impedance as the one you are measuring.

Gain error is determined by the ratio of the cable capacitance to the bias tee capacitance, which should not change much across the range of frequencies. The S540 system can have individual compensation constants for any requested frequency, in the event that the ratio varies.

Three-terminal capacitance measurements

Three-terminal capacitance measurements (C_{iss}, C_{oss}, and C_{rss}) are some of the most challenging, yet commonly used measurements in power device characterization. Making C_{iss}, C_{oss}, and C_{rss} measurements allows you to evaluate required transistor switching characteristics such as speed, energy, and charge.

This type of measurement historically has been done using bench setups. However, Keithley Instruments has developed a procedure for the S540 system that uses its high-voltage matrix to automate these measurements. Keithley High-Voltage Library (HVLib) and Linear Parametric Test Library (LPTLib) commands are used with the Keithley Test Environment (KTE) Software to automate this procedure.

C_{iss}, C_{oss}, and C_{rss} measurements are typically made in the off state, with gate voltage at 0 V DC and at high drain voltage. A bias tee must be connected to each terminal because varying high-voltage biases are applied to each terminal (gate, drain, and source). This connection configuration differs from the standard two-terminal configuration shown in the figure in Bias tees and compensation in a two-terminal AC model (on page 9-4).

Input (C_{iss}) and output (C_{oss}) capacitances are measured in a similar way. Each of the three terminals (gate, drain, and source) must have an independent DC bias. In the AC frequency domain, two terminals are connected to each other in the high-voltage matrix and impedance is measured against the third terminal.

For example, for C_{iss}, drain voltage is usually high, the source is DC grounded, and the gate voltage ensures the off state of the transistor. Then the gate is AC-tied to the sense (low) terminal of the capacitance meter (CMTRL) and the source pin is AC-tied with the drain to the high (AC-drive) side of the CMTR (CMTRH).

Guard challenge for Crss

C_{rss} capacitance measurement is more difficult than the measurement of C_{iss} or C_{oss}. Impedance measurement is done between the gate and drain with AC guarding at the source pin.

In AC guarding, the guarded pin is held as close as possible to AC ground by providing a low-impedance connection to the AC ground, or by applying an active AC signal to the guarded pin. This ensures minimal AC voltage on the guarded pin. However, the ability to minimize AC voltage on the guarded pin is limited by interconnects, and becomes progressively less effective at high frequencies.

The S540 system guards the source by connecting it directly to ground (GND), which bypasses the bias tees in the system. This technique works sufficiently at 100 kHz, but has reduced accuracy at 1 MHz (see the S540 specifications for details). This only affects the quality of the C_{rss} measurement.

Automated high-voltage C-V measurements

Three-terminal capacitance measurements require careful application of complex connections to the capacitance meter (CMTR), bias tees, and DC instruments. The S540 3-kV high-voltage matrix facilitates software-controlled connections, automating these connections for high-voltage capacitance-voltage (C-V) measurements.

The High-Voltage Library (HVLib) `hvcv_3term` command uses these software-controlled connections to make automated three-terminal capacitance measurements. You can use this command without changes, or you can copy it under a different name and customize it for your application. It can be used to measure C_{iss} and C_{oss} parameters and individual capacitances (C_{rss}). The routine uses system-level compensation when the *`doComp`* parameter is enabled.

The `hvcv_3term` command can do device-level compensation, including most of the known compensation models. For example, to run ShortOpenLoad device-level compensation, you must first collect compensation data from the open, short, and load devices.

High-voltage C-V compensation methods

S540 systems with two capacitance meters (CMTRs) can make high-voltage capacitance-voltage (C-V) measurements using two different compensation methods:

- System-level compensation (on page 9-7)
- Device-level compensation (on page 9-12)

The following topics describe each of these methods. For examples of how these methods can be used to make high-voltage C-V measurements, see High-voltage C-V usage scenarios (on page 9-15).

System-level compensation

System-level compensation applies compensation factors to negate the effect of bias tees in the system. It uses a single, preprogrammed set of compensation factors for any pin combination in the system. System-level compensation factors are set at the factory and stored in the system. They can also be recreated at your site if necessary.

System-level compensation is not as accurate as device-level (user-specified) compensation because it does not account for the subtle pin-to-pin differences caused by probe cards, cabling, or test fixtures in the system.

To use this level of compensation, set the *doComp* parameter to `1` or `2` in the `hvcv_test`, `hvcv_sweep`, or `hvcv_3term` high-voltage library (HVLib) commands.

System-level compensation factors are stored in the `cvCALsystem.ini` file. If system-level compensation is enabled, the `hvcv_test`, `hvcv_sweep`, or `hvcv_3term` commands use `hvcv_intcg` function instead of the standard Linear Parametric Test Library (LPTLib) `intgcg` function.

The `hvcv_intcg` command uses one of two different sets of compensation factors:

- When the *doComp* parameter is set to `1`, the `hvcv_intcg` command uses compensation factors stored in the `cvCALsystem.ini` file.
- When the *doComp* parameter is set to `2`, the `hvcv_intcg` command uses compensation factors stored in the `cvCAL.ini` file. This setting does run-time ShortOpenLoad compensation of the raw data.

Procedure overview

Following is a summary of what happens during system-level compensation:

- Preprogrammed compensation factors are retrieved, or new compensation factors can be created at run-time using a custom fixture and load standard (probe card with connection for the discrete capacitors). The default procedure uses preprogrammed compensation factors.
- Compensation factors are stored in the `opt/kiS530/cvCAL.ini` file.
- The `hvcv_intgcg` command reads the compensation factors, and if appropriate, does ShortOpenLoad compensation.

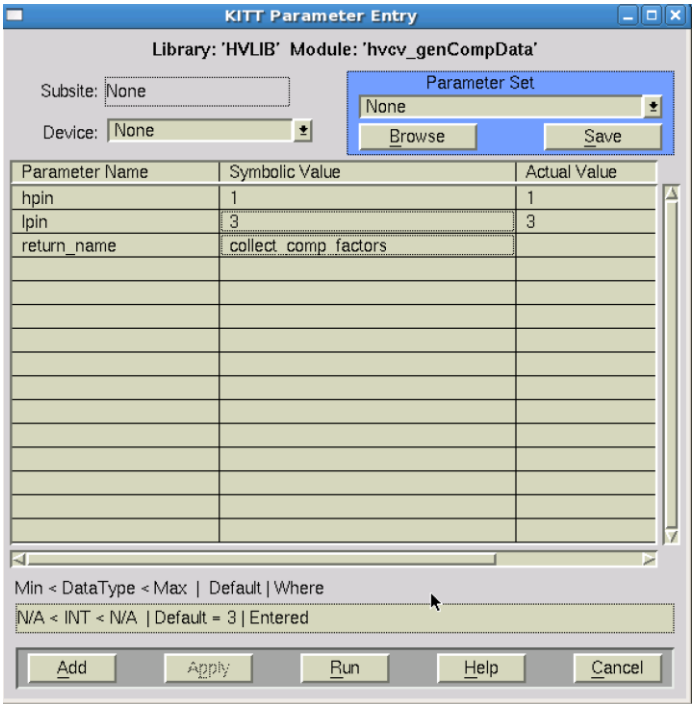
System-level compensation can be enabled or disabled by setting the *`doComp`* parameter to `1` or `0` in the following commands: `hvcv_test`, `hv_sweep`, and `hvcv_3term`.

For examples of specific usage scenarios, see High-voltage C-V usage scenarios (on page 9-15).

Recreating system-level compensation factors

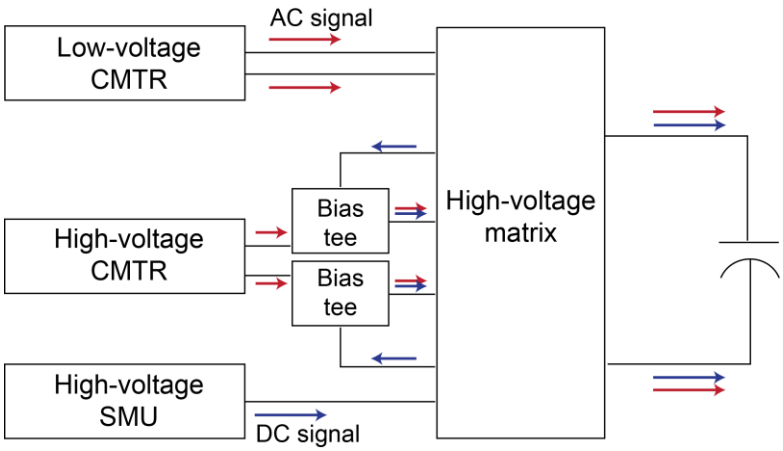
To recreate system-level compensation factors, use the `hvcv_genCompData` command. This command prompts you to connect open, short, and load devices to a selected pin pair. You must use a custom fixture or probe card that allows you to insert a load standard and short device. The following figure shows the selected pin-pair of pin 1 and pin 3. For a load-device or load standard, Keithley recommends using a discrete capacitor with a range of 100 pF to 1 nF.

Figure 217: Compensation data collection function: hvcv_genCompData



System-level compensation uses the algorithm shown in the following figure. If the S540 system has a high-voltage capacitance-voltage (C-V) configuration (high-voltage capacitance meter (CMTR) with bias tees), it also has a low-voltage CMTR, as shown in the following figure.

Figure 218: Using a low-voltage CMTR for high-voltage compensation



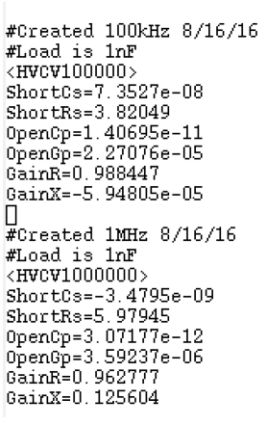
The high-voltage CMTR is connected to the high-voltage matrix through bias tees, which allows a DC bias voltage of up to 3 kV applied to the device.

The low-voltage CMTR is connected directly to high-voltage matrix with no bias-tees and does not exhibit AC signal degradation caused by bias-tees. The low-voltage CMTR is protected from high voltages by protection modules (not shown in the previous figure).

When the `hvcv_genCompData` command is collecting compensation data, the high-voltage and low-voltage CMTRs characterize the open, short, and load devices. The low-voltage CMTR makes the measurements and calculates compensation factors for frequencies from 1e4 to 2e6.

The `hvcv_genCompData` routine then writes the compensation factors for each frequency (10 kHz to 2 MHz) to the `opt/kiS530/cvCAL.ini` file. The following figure shows an example of the compensation factors for the 100 KHz and 1 MHz frequencies. In this figure, `ShortCs` and `ShortRs` characterize the impedance of the short. Short resistance (`ShortRs`) should be below 10 Ω. Values for `OpenCp` are usually below 10 pF. Gain compensation factors (`GainR` and `GainX`) are real and imaginary (reactive) components of the load compensation. `GainX` is usually 0.10 or below, and `GainR` is close to 1.00 (from 0.95 to 1.10). Note that units of `GainX` and `GainR` are dimensionless.

Figure 219: Open, short, and load compensation factors in cvCAL.ini



```
#Created 100kHz 8/16/16
#Load is 1nF
<HVCV100000>
ShortCs=7.3527e-08
ShortRs=3.82049
OpenCp=1.40695e-11
OpenGp=2.27076e-05
GainR=0.988447
GainX=-5.94805e-05

#Created 1MHz 8/16/16
#Load is 1nF
<HVCV1000000>
ShortCs=-3.4795e-09
ShortRs=5.97945
OpenCp=3.07177e-12
OpenGp=3.59237e-06
GainR=0.962777
GainX=0.125604
```

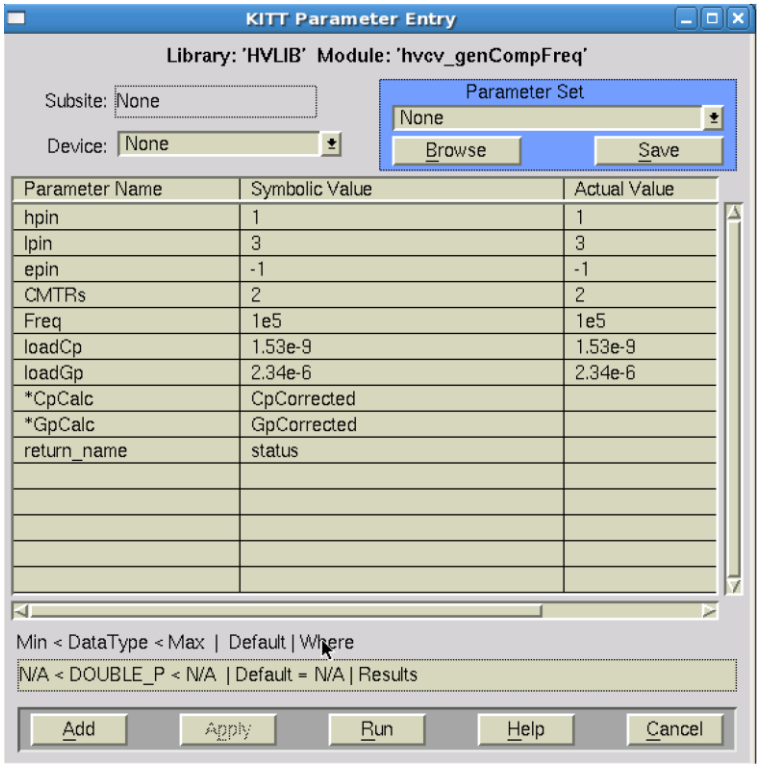
Generating compensation factors for a single frequency

The S540 High-Voltage Library (HVLib) `hvcv_genCompFreq` command generates compensation factors for a single frequency. You can use this command to debug the compensation algorithms, and it prompts you to insert the open, short, and load devices when appropriate.

The `hvcv_genCompFreq` command has two modes of operation based on the number of capacitance meters (CMTRs) that are available in the system. If the *`CMTRs`* parameter is set to `1`, the routine only uses the high-voltage CMTR and uses values specified by the *`loadCP`* and *`loadGP`* parameters for the load device. If the *`CMTRs`* parameter is set to `2`, the second CMTR (low-voltage CMTRL) is used to create reference load data and provided data for the load is ignored.

The following figure shows example parameters for the command; definitions of the parameters follow the figure.

Figure 220: Compensation data collection function: hvcv_genComp_Freq



This example uses the following parameters:

- *hpin*: Pin connected to the capacitance meter high-side (for low-voltage CMTR1H and high-voltage CMTR2H)
- *lpin*: Pins connected to the CMTR low-side (for low-voltage CMTR1L and high-voltage CMTR2L)
- epin: Extra pin connected to CMTR high-voltage guard terminal (CMTR2G)
- *CMTRs*: The number of CMTRs to use for compensation measurements. If the number is 2, the low-voltage CMTR and the high-voltage CMTR are used. If the number is 1, only the high-voltage CMTR is used, and the *loadCp* and loadGp are used for the load.
- *Freq*: Specified frequency
- *loadCp* and *loadGp*: Independently known values of the load device, expressed as capacitance and conductance, according to the parallel model representation

- *CpCalc* and *GpCalc*: Values of the load device after measurement and compensation; these values should be very close to the loadCp and loadGp values when the value of the *CMTRs* parameter is 1
- *Return_name*: Status of the measurement; negative value for failure

Device-level compensation

Device-level compensation applies compensation factors to negate the effect of bias tees, cabling, probe cards, or test fixtures in the system. You create the compensation factors just before testing a device. This results in more accurate high-voltage capacitance-voltage (C-V) measurements.

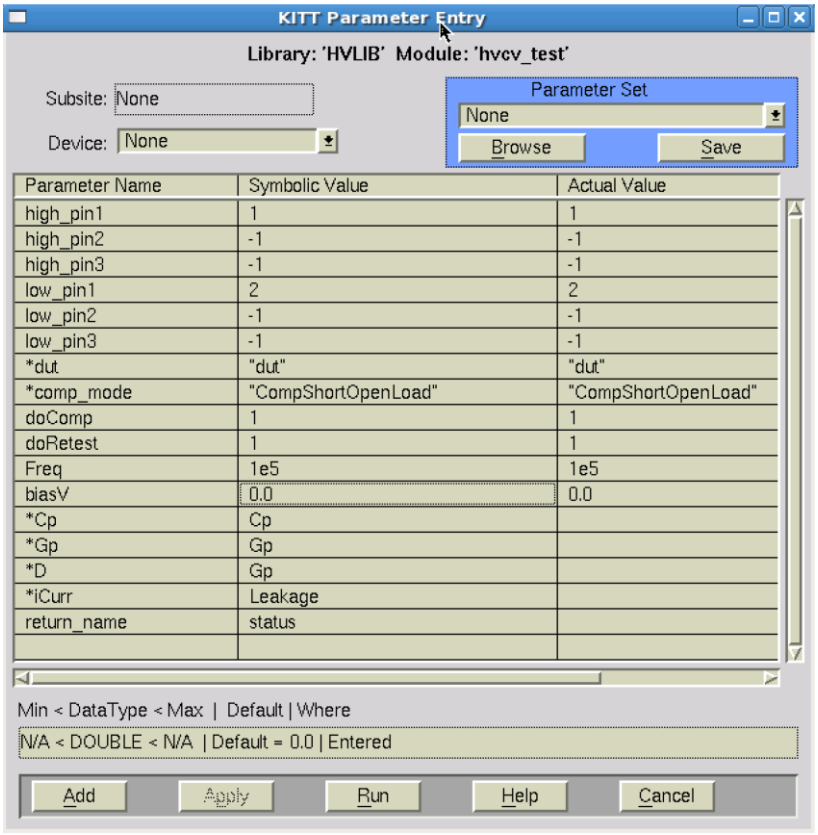
There are several ways to acquire device-level of compensation factors, discussed in the following topics. For examples of specific usage scenarios, see High-voltage C-V usage scenarios (on page 9-15).

Device-level compensation allows you to do compensation for each individual pin-pair at run time during automated testing on the wafer. Available compensation methods are Open, Short, Load, OpenLoad, ShortOpen, and ShortOpenLoad.

Open measurement is done when the chuck is down. Short measurements can be made on any metal surface or connected pads on the wafer.

Selection of the known load device on the wafer can be difficult because it requires C-V characterization of the capacitor on the wafer with no bias tee connection. If your S540 system is configured with a high-voltage capacitance meter (CMTR) and a low-voltage CMTR, you can use the low-voltage CMTR (CMTR1) to measure the expected value of the load.

Figure 221: Example of the hvcv_test in KITT



Procedure overview

The S540 High-Voltage Library (HVLib) contains several device-level commands that you can configure for capacitance measurement:

- `hvcv_test`: Makes a two-terminal single DC bias measurement (the figure in Device-level compensation (on page 9-12) shows example settings for this command)
- `hvcv_sweep`: Collects capacitance-voltage (C-V) data and does a DC bias sweep
- `hvcv_3term`: Measures output capacitance (C_{oss}), input capacitance (C_{iss}), and short-circuit reverse transfer capacitance, common source (C_{rss}) of three-terminal devices

These device-level, user-configured commands are structured similarly to allow for device-level compensation.

The following example of device-level ShortOpenLoad compensation shows how device-level compensation can improve data accuracy. This is useful in situations where impedance of the device is small (large capacitances) or for larger frequencies (for example, 1 MHz).

This example makes a two-terminal capacitance measurement and does full ShortOpenLoad compensation using the `hvcv_test` command.

1. Select a short device with pins 1 and 2 connected.
2. Execute the following test:

```
doComp = 1;
doRetest = 0;
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "short", "CompNone", doComp,
doRetest, Freq, DcBias, Cp, Gp, Gp, Leakage);
```

3. On the load device (select a device with an impedance close to the tested impedance), execute the following test:

```
doComp = 1;
doRetest = 0;
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "load", "CompNone", doComp,
doRetest, Freq, DcBias, Cp, Gp, Gp, Leakage);
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "loadEx", "CompNone", doComp,
doRetest, Freq, DcBias, Cp, Gp, Gp, Leakage);
```

4. On the device under test (DUT), execute the following test:

```
doComp = 1;
doRetest = 0;
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "open", "CompNone", doComp,
doRetest, Freq, DcBias, Cp, Gp, Gp, Leakage);
status = hvcv_test(pin1, -1, -1, pin2, -1, -1, "dut", "CompShortOpenLoad",
doComp, doRetest, Freq, DcBias, Cp, Gp, Gp, Leakage);
```

The sequence of tests shown above does the following:

- Enables all measurements to run through system-level compensation one time (*doComp* parameter set to `1`).
- Forces all compensation data (`short`, `open`, `load`, and `loadEx`) to be collected only once (*doRetest* parameter set to `0`). To force data to be collected again, you can set the *doRetest* parameter to `1`.
- Sets the *comp_mode* parameter to `CompShortOpenLoad` to do ShortOpenLoad compensation.
- Sets the *dut* parameter to `short` to collect data on the short device. This type of compensation is useful when there is very large capacitance with small impedance.
- Sets the *dut* parameter to `load` to collect gain compensation data with the high-voltage CMTR.
- Sets the *dut* parameter to `loadEx` to collect gain compensation data with the low-voltage CMTR.
- Sets the *dut* parameter to `open` to move the chuck down and collect compensation data.

- Enables characterization of the device under test (DUT) for short, open, and load devices by setting the *comp_mode* parameter to `CompShortOpenLoad`. This step uses previously collected data; if short, open, or load data was not previously collected, the test fails (even if the appropriate compensation mode was specified).
- Stores compensation data in memory; this data is only available within the same process. This means that if, for example, lines of code are executed one at a time using the Keithley Interactive Test Tool (KITT), compensation data will not be available and the test will fail. For this debug scenario, use the `hvcv_storeData` command to store required data in the data pool.

High-voltage C-V usage scenarios

The following topics contain example high-voltage capacitance-voltage C-V measurement applications using system-level or device-level compensation factors.

Two-terminal HV C-V measurement with system-level compensation

This is the simplest type of high-voltage capacitance-voltage (C-V) measurement. One of the following S540 High-Voltage Library (HVLib) commands is used:

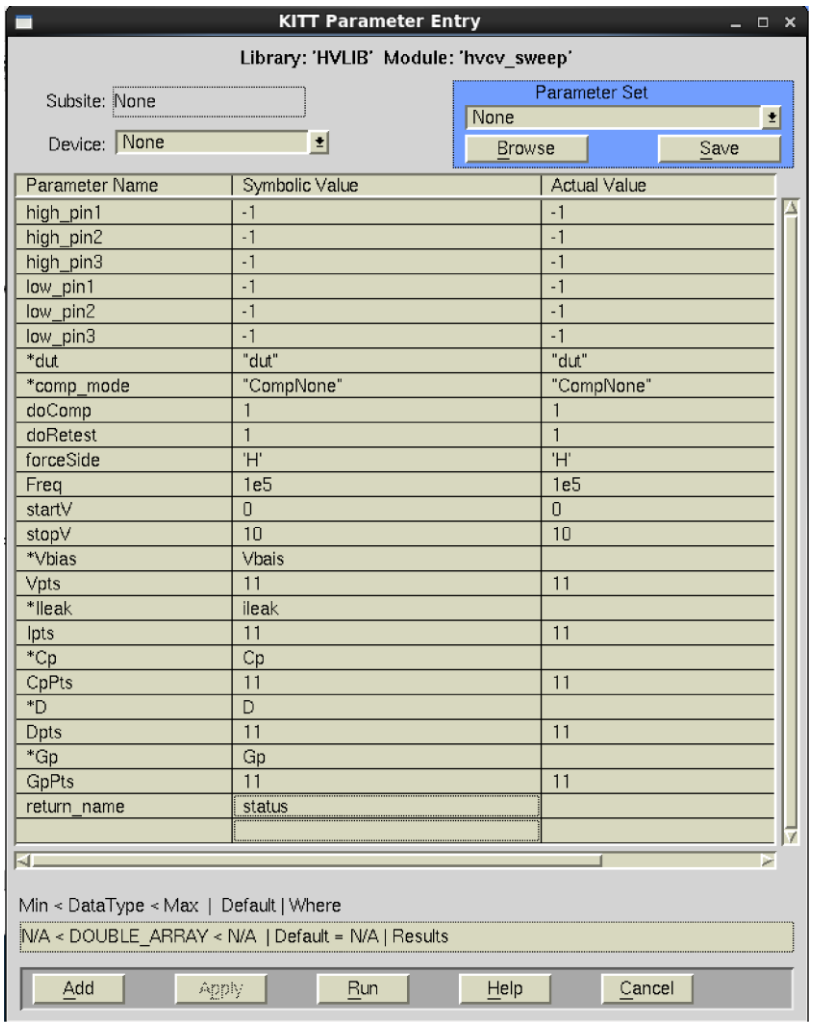
- `hvcv_test`: A single measurement is made at one voltage bias level
- `hvcv_sweep`: An array of measurements is made using an array of voltage biases

The following code is an example of using the `hvcv_sweep` command:

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "dut", "CompNone", 1, "H", 0, 1e5, 0,
   10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

In the example above, `"dut"` indicates that the test will run on the device under test. `"CompNone"` indicates that device-level compensation will not be used. The value following `"CompNone"` (the *doComp* parameter) is set to `1` to specify that the test uses system-level compensation factors supplied by the factory.

Figure 222: High-voltage C-V sweep test



Two-terminal HV C-V measurement with device-level compensation

This example shows how you can run bench-top tests on a single device using the Keithley Interactive Test Tool (KITT). There are two parts to this process:

1. Run device-level compensation using the `hvcv_genCompData` user module in KITT:
   a. Input the system pin numbers to use to test your device.
   b. Run the module.
   c. Following the prompts in KITT, connect the pins to an open device, a short device, and a load device. The load device should be a known good device with capacitance properties similar to the ones you expect from the device you intend to test.
      The S540 system uses both capacitance meters (high-voltage and low-voltage CMTRs) in the system to measure this device with and without connections through bias tees.
      When the test is complete, compensation data is stored to a file on the system for later retrieval.
2. Make the measurement using either the `hvcv_test` command (for a single measurement at one voltage bias level) or the `hvcv_sweep` command (for an array of measurements using an array of voltage biases).

The following code is an example using the `hvcv_sweep` command.

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "dut", "CompNone", 2, "H", 0, 1e5, 0,
    10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

In the example above, `"dut"` indicates that the test will run on the device under test. `"CompNone"` indicates that automated device-level compensation will not be used. The value following `"CompNone"` (the *doComp* parameter) is set to `2` to specify that the test uses the device-level compensation data that you just created using the `hvcv_genCompData` module in step 1.

NOTE

The compensation factors that you created using the `hvcv_genCompData` module contain data for all frequencies available on the system at the time that you ran the module. They are specific to the pins and setup configured when the module was run. These compensation factors remain on the system for reuse until the `hvcv_genCompData` module runs again. When the module runs again, the data is overwritten with new compensation factors based on the configuration at that time.

Automated two-terminal HV C-V measurement with device-level compensation

If you are using an automated test plan module to test a series of devices, you start by collecting device-level compensation data. The process of doing this is slightly different than testing at the bench level.

In an automated setting, you can choose any combination of the following types of device-level compensation:

- Open
- Short
- Load

Each type of device-level compensation data is collected separately using one of the following S540 High-Voltage Library (HVLib) commands:

- `hvcv_test`: A single measurement is made at one voltage bias level
- `hvcv_sweep`: An array of measurements is made using an array of voltage biases

The following code is an example of each type of device-level compensation using the `hvcv_sweep` command.

Open compensation:

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "open", "CompNone", 0, "H", 0, 1e5, 0,
   10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

Short compensation:

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "short", "CompNone", 0, "H", 0, 1e5, 0,
   10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

Load compensation:

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "load", "CompNone", 0, "H", 0, 1e5, 0,
   10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "loadEX", "CompNone", 0, "H", 0, 1e5,
   0, 10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

In each of the examples above, the *`dut`* parameter specifies the type of compensation device to be used (`open`, `short`, or `load`).

NOTE

For the load type of compensation, you must run the test twice: Once using *dut* parameter `load` and once using the *dut* parameter `loadEx`. This tells the system to run the test first using the high-voltage capacitance meter (CMTR) connected through bias tees, then to run the test using the low-voltage CMTR with no connection through bias tees. The same parameters should be used for both the `load` and `loadEx` DUT tests.

In the examples above, `"CompNone"` indicates that no compensation is done as you are collecting compensation data. The value following `"CompNone"` (the *doComp* parameter) is set to 0 so that the routine does not do ShortOpenLoad compensation as you are collecting compensation data.

The *doRetest* parameter is set to `0` so that if the `hvcv_sweep` command is called again during the same run of the test plan module (with the identical pin configuration), the retest is skipped. This saves time by using compensation data that was previously collected and stored in the data pool. This is useful when you are testing multiple devices or wafers. If you want to collect new compensation data for each pin configuration (even if it is identical to a previous pin configuration), set the *doRetest* parameter to `1`.

NOTE

Use prober commands to ensure that the pins are connected to an appropriate device (or not connected to anything if the open compensation mode is selected) before each of the compensation commands is run. See the *S530/S540 Prober and Prober Driver Manual* (part number S530-911-01) for descriptions of prober commands.

Once compensation data has been collected, measure the device using either the `hvcv_test` command (for a single measurement at one voltage bias level), or the `hvcv_sweep` command (for an array of measurements using an array of voltage biases). Following is an example using the `hvcv_sweep` command.

```
return_name = hvcv_sweep(1, -1, -1, 2, -1, -1, "dut", "CompShortOpenLoad", 0, "H", 0,
   1e5, 0, 10, Vbias, 11, ILeak, 11, Cp, 11, D, 11, Gp, 11)
```

In this example, `"dut"` indicates that you are testing the device under test. `"CompShortOpenLoad"` specifies that all three types of compensation are used. Alternatively, you could use `"CompShortOpen"` to use short and open compensation only, or `"CompOpen"` to use only open compensation, and so on. The *doComp* parameter is set to `0` so that automated device-level compensation is used instead of system-level or bench-level compensation.

Three-terminal HV C-V measurement with device-level compensation

There are several steps that must be completed in a single test run for this scenario:

- Collect device-level compensation factors
- Store compensation factors in the data pool
- Test the device

Following is an example of three-terminal high-voltage capacitance-voltage (C-V) measurement using device-level compensation.

Step 1: Collect device-level compensation factors

Do the device-level compensation for the type of devices you are testing (open, short, or load devices, or any combination of those devices) using the `hvcv_3term` module in the Keithley Interactive Test Tool (KITT). For example:

Open compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "open", "CompNone", 1e5, 0, 0, drainV,
    11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

Short compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "short", "CompNone", 1e5, 0, 0,
    drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

Load compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "load", "CompNone", 1e5, 0, 0, drainV,
    11, drainI, 11, Cp, 11, D, 11, Gp, 11)
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "loadEx", "CompNone", 1e5, 0, 0,
    drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

In each of the examples above, the `dut` parameter indicates the type of compensation device to use (open, short, or load).

NOTE

For the load type of compensation, you must run the test twice: Once using *`dut`* parameter `load` and once using the *`dut`* parameter `loadEx`. This tells the system to run the test first using the high-voltage capacitance meter (CMTR) connected through bias tees, then to run the test using the low-voltage CMTR with no connection through bias tees. The same parameters should be used for both the `load` and `loadEx` DUT tests.

"CompNone" indicates that no compensation is done as you are collecting compensation data. When the module is run in this mode, only one actual value is returned for Cp and Gp, and all the other values in the array are 0. This is because compensation is run at a 0 V bias only. Record the Cp and Gp values returned. These are the compensation factors that you use as inputs in step 2.

When this step is complete, record these values.

Step 2: Store compensation factors in the system data pool

Use the hvcv_storeData command as shown below to store compensation factors in the data pool.

Open compensation:

```
return_name = hvcv_storeData("D1_G4_S5_Mode:Ciss", "open", 1e5,3.85523e-12,
   -4.50236e-8)
```

Short compensation:

```
return_name = hvcv_storeData("D1_G4_S5_Mode:Ciss", "short", 1e5,8.3487E-8, 0.0136743)
```

Load compensation:

```
return_name = hvcv_storeData("D1_G4_S5_Mode:Ciss", "load", 1e5, 9.61043e-10,
   2.87691e-6)
return_name = hvcv_storeData("D1_G4_S5_Mode:Ciss", "loadEx", 1e5, 9.61043e-10,
   2.87691e-6)
```

The value of the *label* parameter in each of the code examples above is determined by the compensation mode you are using (Ciss, Coss, or Crss) and the pin numbers used to test the device. In this example, the label parameter is "D1_G4_S5_Mode:Ciss", which means the compensation mode is C_{iss}, the drain is connected to pin 1, the gate is connected to pin 4, and the source is connected to pin 5.

Step 3: Run the hvcv_3term module in KITT

Immediately after storing the compensation factors in the data pool, run the hvcv_3term module in KITT to test the device.

NOTE

The data pool only retains information for the duration of a single test run in KITT. Because of this, you must run all hvcv_storeData modules and the hvcv_3term module in a single KITT test run.

The following is an example of how to complete this step.

```
return_name1 = hvcv_storeData("D1_G4_S5_Mode:Ciss", "open", 1e5,3.85523e-12,
   -4.50236e-8)
return_name2 = hvcv_storeData("D1_G4_S5_Mode:Ciss", "short", 1e5,8.3487E-8,
   0.0136743)
return_name3 = hvcv_storeData("D1_G4_S5_Mode:Ciss", "load", 1e5, 9.61043e-10,
   2.87691e-6)
return_name4 = hvcv_storeData("D1_G4_S5_Mode:Ciss", "loadEx", 1e5, 9.61043e-10,
   2.87691e-6)
return_name5 = hvcv_3term(1, 4, 5, 0, 0, 10, "Ciss", "dut", "CompShortOpenLoad", 1e5,
   0, 1, V, 11, I, 11, Cp, 11, D, 11, Gp, 11)
```

In the `hvcv_3term` example above, `"dut"` indicates that the test will run on the device under test. `"CompShortOpenLoad"` specifies that all three modes of compensation will be used. You could instead use any of the other compensation modes, for example: `"CompShortOpen"` for short and open only, or `"CompOpen"` for open only. The `doComp` parameter is set to `0` to specify that automated device-level compensation is used instead of system-level compensation.

Automated three-terminal HV C-V measurement with device-level compensation

This type of testing is similar to automated two-terminal testing, except instead of using the `hvcv_test` or `hvcv_sweep` commands, you use the `hvcv_3term` command.

Open compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "open", "CompNone", 1e5, 0, 0, drainV,
   11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

Short compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "short", "CompNone", 1e5, 0, 0,
   drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

Load compensation:

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "load", "CompNone", 1e5, 0, 0, drainV,
   11, drainI, 11, Cp, 11, D, 11, Gp, 11)
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "loadEx", "CompNone", 1e5, 0, 0,
   drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

In each of the examples above, the *`dut`* parameter specifies the type of compensation device to be used (`open`, `short`, or `load`).

NOTE

For the load type of compensation, you must run the test twice: Once using *dut* parameter `load` and once using the *dut* parameter `loadEx`. This tells the system to run the test first using the high-voltage capacitance meter (CMTR) connected through bias tees, then to run the test using the low-voltage CMTR with no connection through bias tees. The same parameters should be used for both the `load` and `loadEx` DUT tests.

`"CompNone"` indicates that no compensation is done as you are collecting compensation data. The value following `"CompNone"` (the *doComp* parameter) is set to 0 so that the routine does not do ShortOpenLoad compensation as you are collecting compensation data.

The *doRetest* parameter is set to `0` so that if the `hvcv_3term` command is called again during the same run of the test plan module (with the identical pin configuration), the retest is skipped. This saves time by using compensation data that was previously collected and stored in the data pool. This is useful when you are testing multiple devices or wafers. If you want to collect new compensation data for each pin configuration (even if it is identical to a previous pin configuration), set the *doRetest* parameter to `1`.

NOTE

Use prober commands to ensure that the pins are connected to an appropriate device (or not connected to anything if the open compensation mode is selected) before each of the compensation commands is run. See the *S530/S540 Prober and Prober Driver Manual* (part number S530-911-01) for descriptions of prober commands.

Once compensation data has been collected, measure the device using the `hvcv_3term` command, as shown below.

```
return_name = hvcv_3term(1, 2, 3, 0, 0, 10, "Ciss", "dut", "CompShortOpenLoad", 1e5,
    0, 0, drainV, 11, drainI, 11, Cp, 11, D, 11, Gp, 11)
```

In this example, `"dut"` indicates that you are testing the device under test. `"CompShortOpenLoad"` specifies that all three types of compensation are used. Alternatively, you could use `"CompShortOpen"` to use short and open compensation only, or `"CompOpen"` to use only open compensation, and so on. The *doComp* parameter is set to `0` so that automated device-level compensation is used instead of system-level or bench-level compensation.

High-Voltage Library (HVLib) commands

The following table contains the High-Voltage Library (HVLib) commands and a brief description of each. For detailed descriptions of each of the commands, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`gate_charge`	Measures the gate charge required to switch on the power transistor.
`hv_bvsweep`	Does a breakdown voltage sweep.
`hvcv_3term`	Measures output capacitance, input capacitance, or short-circuit reverse transfer capacitance of three-terminal devices.
`hvcv_3term_basic`	Makes basic measurements of output capacitance, input capacitance, or short-circuit reverse transfer capacitance of three-terminal devices.
`hvcv_comp`	Does complex mathematical calculations to implement specified impedance compensation models.
`hvcv_genCompData`	Generates correction factors for system-level high-voltage capacitance-voltage (C-V) compensation.
`hvcv_genCompFreq`	Generates compensation factors for system-level capacitance compensation for a single, specified frequency.
`hvcv_getData`	Gets compensated capacitance and compensated conductance data from the data pool.
`hvcv_intgcg`	Measures capacitance and does system-level ShortOpenLoad compensation on the high-voltage capacitance meter (CMTR).
`hvcv_measure`	Measures and stores compensated capacitance and compensated conductance values.
`hvcv_storeData`	Stores compensated capacitance and compensated conductance data in the data pool.
`hvcv_sweep`	Does a high-voltage C-V sweep.
`hvcv_sweep_basic`	Does a basic high-voltage C-V sweep.
`hvcv_test`	Makes a high-voltage C-V measurement at a single frequency.
`hvcv_test_basic`	Makes a basic high-voltage C-V measurement at a single frequency.

Section 10

Frequency analysis

In this section:

Introduction

This section describes the Keithley frequency analysis option. This option consists of an RSA306B USB Spectrum Analyzer (or Model 4200-SCP2HR scope card if you have an older system) that can be used with Linear Parametric Test Library (LPTLib) commands to analyze frequencies and make ring oscillator measurements.

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

In most cases, the frequency analysis option is integrated in your S540 parametric test system at the factory; however, the hardware is field-installable by a Keithley field service engineer (FSE).

Prerequisites

To use the frequency analysis option in your system, you should be familiar with fundamental C programming, as well as with the operation of your Keithley S540 parametric test system. If you are not familiar with this programming language or the test system, become familiar with them before proceeding.

Ring oscillator structures and measurement

The main purpose of ring oscillator structures is to allow chip designers to characterize the switching speed and propagation delays of logic gates. Propagation delay is calculated using the following formula:

Equation 1: Propagation delay formula

$$\frac{1}{2nf}$$

Where: `n` is the number of stages in the CMOS inverter chain

`f` is the measured frequency of the circuit

Using the RSA306B spectrum analyzer (instead of a frequency counter) is necessary because the signal frequency of the ring oscillator is often more than several hundred megahertz. This signal frequency is well beyond the measurement capability of frequency counters when connected through a tester.

Making a measurement

Measurements made using the RSA306B spectrum analyzer (or scope card on older systems) require the following basic sequence of events: Initialize, setup, and acquire.

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

To take a measurement using the basic sequence of required events:

NOTE

To use the spectrum analyzer (or scope card on older systems) when creating a Keithley Interactive Test Tool (KITT) macro, open the Linear Parametric Test Library (LPTLib) using the KITT libraries menu.

1. Initialize the spectrum analyzer or scope card using the `rsa_init` (if you have spectrum analyzer) or `scp_init` (if you have a scope card) command.
2. Set up the measurement conditions using the `rsa_setup` or `scp_setup` command.
3. Make the measurement using one of the measurement commands (see the following figures).

Figure 223: Using KITT to make a scope card measurement

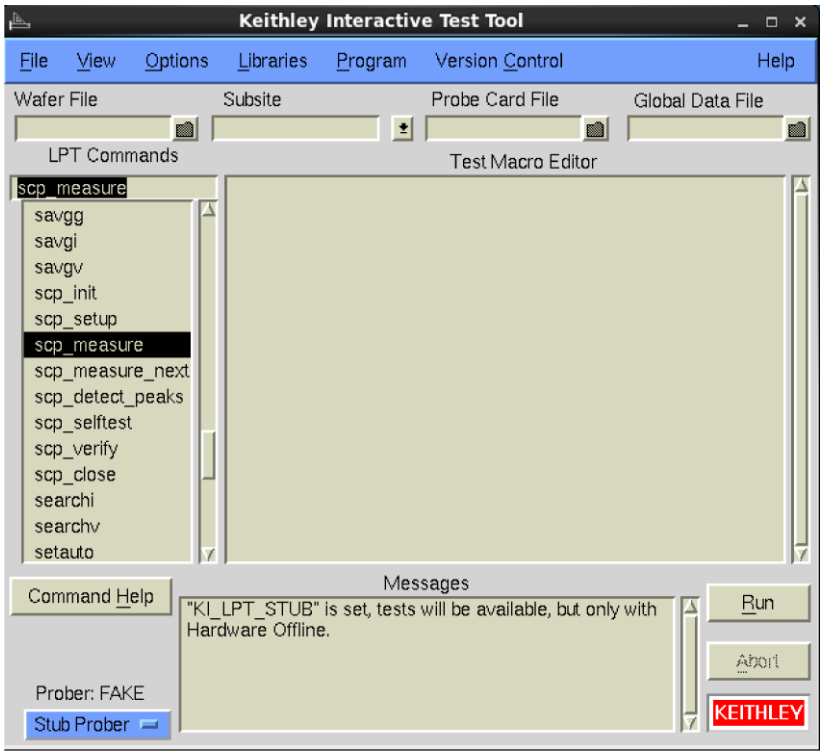
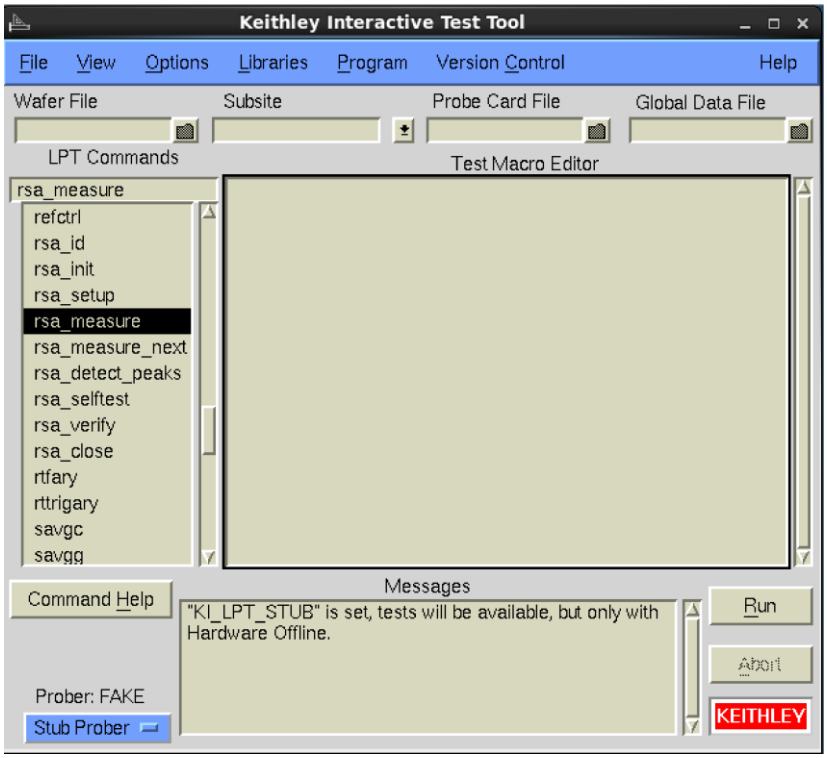


Figure 224: Using KITT to make a measurement with the RSA306B

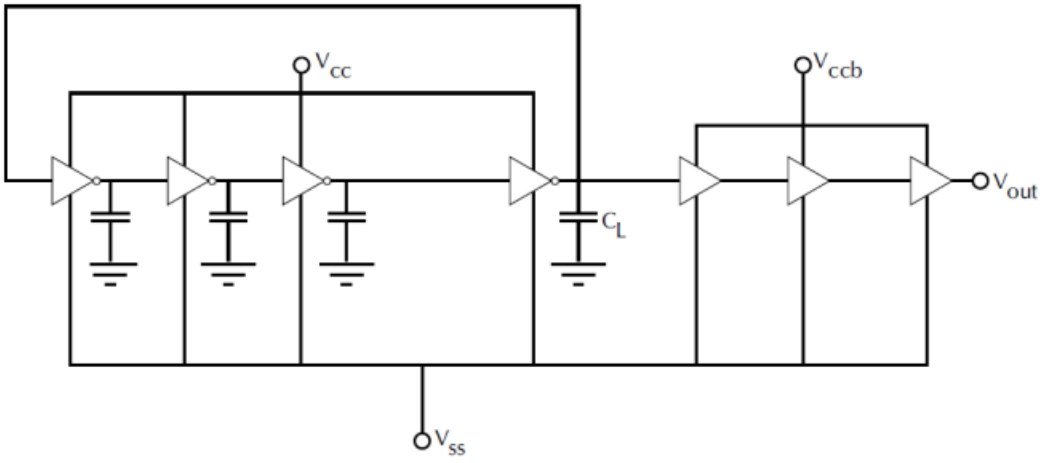


Measurement commands for ring oscillator applications

The measurement commands for ring oscillator measurement are C-language commands that measure the following signal characteristics:

- Inverter chain CMOS ring oscillators (see the following figure)
- Differential ring oscillators (supported types only)

Figure 225: Generalized inverter chain ring



RSA306B LPTLib commands

NOTE

You can use the following Linear Parametric Test Library (LPTLib) commands to make measurements with the S540 only if you have an RSA306B USB Spectrum Analyzer in your system. In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Command	Description
`rsa_close`	Disconnect communications to the signal analyzer
`rsa_detect_peaks`	Return frequencies in signal amplitude order
`rsa_init`	Initialize the signal analyzer to its default state
`rsa_measure`	Measure the frequency and amplitude of the strongest signal
`rsa_measure_next`	Return the frequency and amplitude of the next highest peak in the frequency spectrum
`rsa_selftest`	Do a self-test of the signal analyzer
`rsa_setup`	Set the start, stop, and step frequencies of a scan

Scope LPTLib commands

NOTE

You can use the following LPTLib commands to make measurements with the S540 only if you have a 4200-SCP2HR scope card in your system.

For more detailed information about these commands, see the "LPTLib command reference" section in the *Keithley Test Environment (KTE) Programmer's Manual* (part number S500-904-01).

Command	Description
`scp_close`	Disconnect communications to the scope card.
`scp_detect_peaks`	Return frequencies in signal amplitude order.
`scp_init`	Initialize the scope card to a default state.
`scp_measure`	Measure frequency and amplitude of the strongest signal.
`scp_measure_next`	Get the frequency and amplitude of next highest peak in frequency spectrum.
`scp_selftest`	Run an internal self-test of the scope card.
`scp_setup`	Set the start, stop, and step frequencies of a scan.

Create a KITT macro

The following code creates a KITT macro for the S540 that you can use to make a spectrum analyzer or scope card measurement.

NOTE

In Keithley systems, the RSA306B functions as a replacement for discontinued scope cards. Spectrum analyzer capabilities may be added in the future.

Spectrum analyzer macro

```
double vcc = 5.0

conpin(SMU1, vcc_pin, vccb_pin, 0)
conpin(vss_pin,GND, 0)
conpin(RSA1A, out_pin, 0)
forcev(SMU1, vcc)

status_1 = rsa_init()
status_2 = rsa_setup(0e6, 15e6, 1e6)
status_3 = rsa_measure(frequency,level)
```

Scope card macro

```
double vcc = 5.0

conpin(SMU1, vcc_pin, vccb_pin, 0)
conpin(vss_pin,GND, 0)
conpin(SCP1A, out_pin, 0)
forcev(SMU1, vcc)

status_1 = scp_init()
status_2 = scp_setup(0e6, 15e6, 1e6)
status_3 = scp_measure(frequency,level)
```

Section 11

Pulse generation

In this section:

Introduction

Keithley's S540 pulse generator unit (PGU) option consists of the Model 4220-PGU pulse card and a control software library called KIPulse. The control software library is integrated in the user library of the Keithley parametric tester. In most cases, your pulse generator will be integrated at the factory. However, the hardware is field-installable.

NOTE

The S540 supports up to three 4220-PGU.

The S540 pulse generator unit option includes the following:

- Model 4220-PGU pulse card
- Customer documentation
- Control software appropriate for your particular test system and computer platform

To use the 4220-PGU in your system, you should be familiar with fundamental C programming, as well as with the operation of your Keithley S540 parametric test system. If you are not familiar with this programming language or the test system, become familiar with them before proceeding.

Pulse measurement considerations

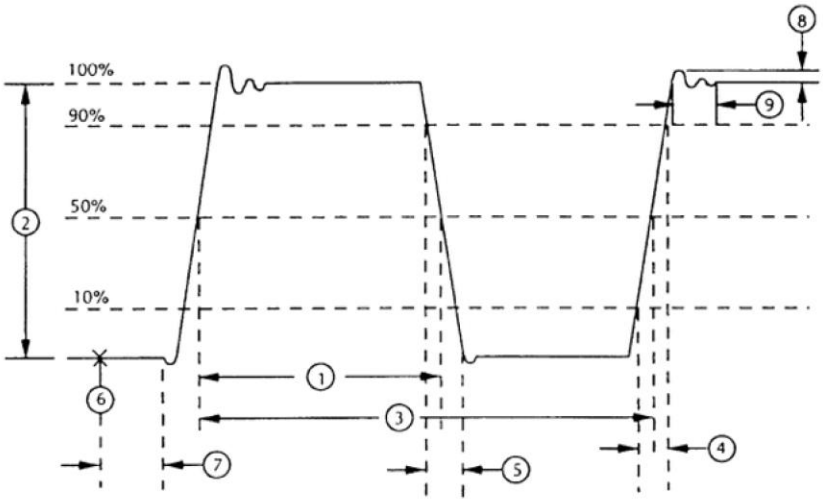
The following information will help you use the pulse generator unit (PGU) option with your S540 system.

- **Pulse width:** Pulse width is measured at the 50% level of the leading edge to the 50% point of the trailing edge. This is expressed in seconds.
- **Pulse height:** The pulse height is measured from the baseline of the pulse to the 100% point and is expressed in volts.
- **Period:** The period of a burst of pulses is the time between the 50% point of two consecutive leading edges of the pulse stream and is expressed in seconds.
- **Rise time:** The rise time of a pulse is the time it takes for the leading edge of a pulse to rise from the 10% level to the 90% level and is expressed in seconds.
- **Fall time:** The fall time is the time it takes to fall from the 90% level to the 10% level.
- **Trigger point:** The trigger point of a pulse is the time at which a trigger signal (software or hardware) is applied to the pulse generator. This point is arbitrary and user-dependent.
- **Pulse delay:** The time between the trigger point and the beginning of the pulse leading edge is the pulse delay. It is expressed in seconds.
- **Overshoot:** The distance between the programmed height or 100% mark and the peak height is the overshoot. Overshoot is expressed as a percentage using the following formula:

Figure 226: Overshoot formula

$$\text{overshoot} = \frac{\text{Peak Height} - \text{Programmed Height}}{\text{Programmed Height}} \times 100$$

Figure 227: Waveform parameters



- **Settling time:** The settling time of a pulse waveform is the time it takes the waveform to settle from its peak value to its programmed value and is expressed in seconds.
- **Offset:** The offset is the programmed voltage shift of the 50% point of the pulse from the 0 V baseline and is expressed in volts.

Command summary

The following table contains brief descriptions of the high-level software commands in the control software. For more detailed information, see the *Keithley Test Environment (KTE) Programmer's Manual* (part number S540-904-01).

NOTE

These commands are only compatible with systems that have 4220-PGU pulse cards.

Command	Description
pgu_current_limit	Force a voltage or current.
pgu_delay	Set the trigger delay time.
pgu_fall	Set the fall time of the pulse.
pgu_halt	Stop all the pulse channels.
pgu_height	Set the peak-to-peak height of the pulse.
pgu_init	Initialize communication with pulse card and set pulse generator to default conditions.
pgu_load	Set the load impedance of a pulse.
pgu_mode	Set the pulse mode of the pulse generator.
pgu_offset	Set the peak-to-peak height and DC offset of the pulse.
pgu_period	Set the period of the pulse in seconds.
pgu_range	Set the voltage range of a pulse generator channel.
pgu_rise	Set the rise time of the pulse.
pgu_trig	Trigger first pulse generator unit and output waveforms.
pgu_trig_burst	Trigger a specified number of pulses.
pgu_trig_unit	Trigger a specified pulse generator unit, or units, to output waveforms.
pgu_width	Set the width of the pulse.

Section 12

Diagnostics and troubleshooting

In this section:

Overview

The S540 diagnostic software is a suite of software tests you can use to verify the correct functionality and performance of the instruments in the S540 Power Semiconductor Test System. This section describes how to use the diagnostic software tool to verify that your S540 system is performing correctly.

The diagnostics software is available for both the Keithley Test Environment (KTE) and Automated Characterization Suite (ACS) software on the S540. The system control and output of the diagnostics software is identical for both versions. Where necessary, specific ACS or KTE instructions are provided.

The diagnostics process

NOTE

You must install the blank probe card in the system before running diagnostics. There must be at least two pins configured and connected from the matrix to the probe card through the probe card adapter (PCA). For high-voltage systems, there must be at least three pins configured and connected from the matrix to the probe card through the PCA.

⚠ WARNING

Hazardous voltages may be present on the probe card adapter, even after you disengage the interlock. Cables can retain charges after the interlock is disengaged, exposing you to live voltages that, if contacted, may cause personal injury or death.

Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

Overview of the diagnostics process:

1.
2. Locate and run the diagnostics user interface:
   - In ACS, double-click the diagnostics icon  to run the diagnostics software program, or double-click the `diagnostic.exe` file, located in `C:\ACS\Diagnostics\`.
   - In KTE, the diagnostics program (`diags`) is in `$KIBIN` (default = `/opt/kiS540/bin`) and can be run by typing `diags` at the terminal prompt.
3. After the software starts, prerequisite testing automatically begins. Depending on your system configuration (for example, number of source-measure units (SMUs) and pins), this may take from 5 minutes to 40 minutes.
4. After the prerequisite tests complete and pass, the diagnostics program is ready to run diagnostic tests. Continue to the next step to complete full diagnostic testing on the system.
5. Verify the testing mode is set to **Diagnostics** and that all tests are set to be executed (see User interface (on page 12-3) to see the location of the **Execute** button and the **All** option).
6. Click the **Execute** button (on the upper right side of the interface). Depending on your system configuration (for example, number of SMUs and pins), the full diagnostic test takes from 30 minutes to 180 minutes.
7. When diagnostic testing is complete, the main user interface indicates whether the tests passed or failed. Information about the status and results of all the completed tests is visible in the tabs located at the bottom of the user interface (see the User interface (on page 12-3) topic for an example).

NOTE

When running the diagnostic user interface without command-line arguments as described above, the diagnostic tests run in default mode. This mode allows testing to continue if a failure occurs. For information about other modes, see Command-line arguments (on page 12-12) and Menu and toolbar selections (on page 12-11).

WARNING

The diagnostics and system verification tests output voltage at the output cables and the probe card. Ensure safe operation by properly using safety interlocks, as described in the "Safety interlocks" section of the *S540 Administrative Guide* (part number S540-924-01).

User interface

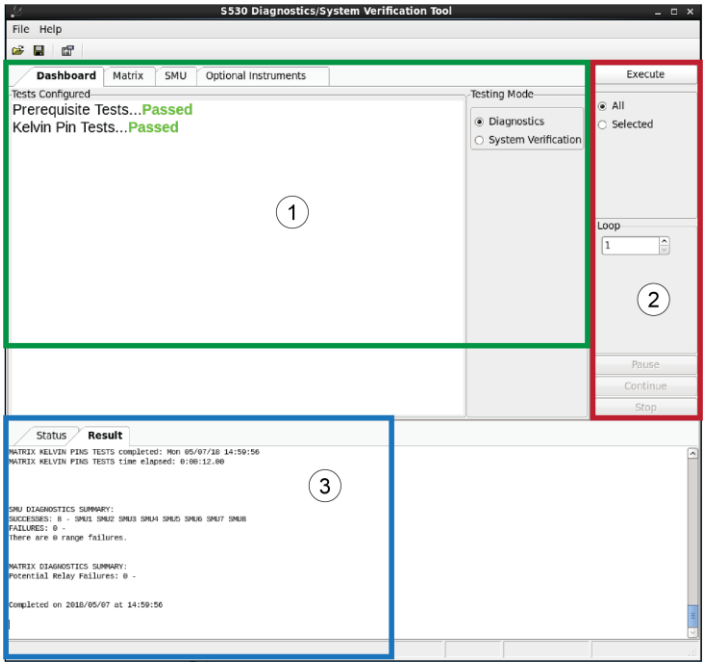
NOTE

The images in this section may look different on your system; available options vary depending on the instruments you have in your system.

The diagnostics software user interface is divided into the following areas (as shown in the following figure):

1. Selection area (on page 12-4)
2. Control area (on page 12-5)
3. Status and Result area (on page 12-6)

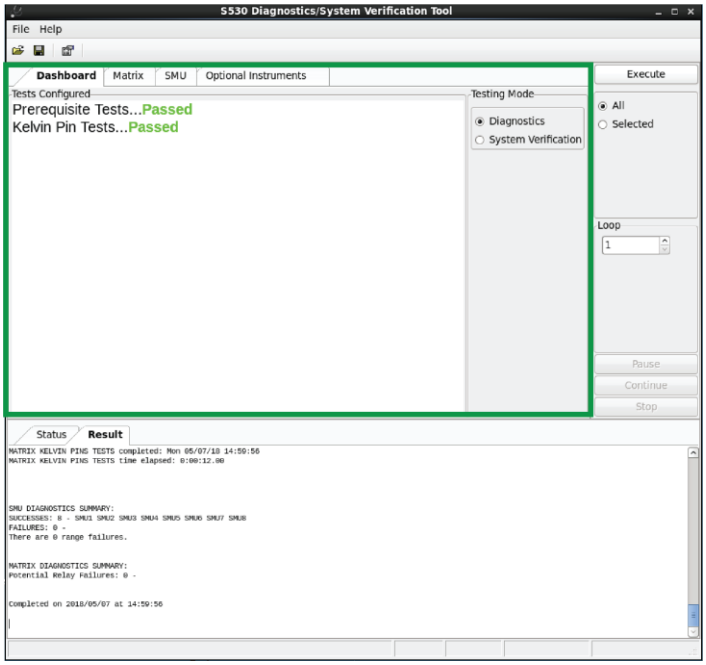
Figure 228: KTE diagnostics user interface



Selection area

The selection area is on the upper left side of the user interface.

Figure 229: Selection area



The selection area contains the following tabs:

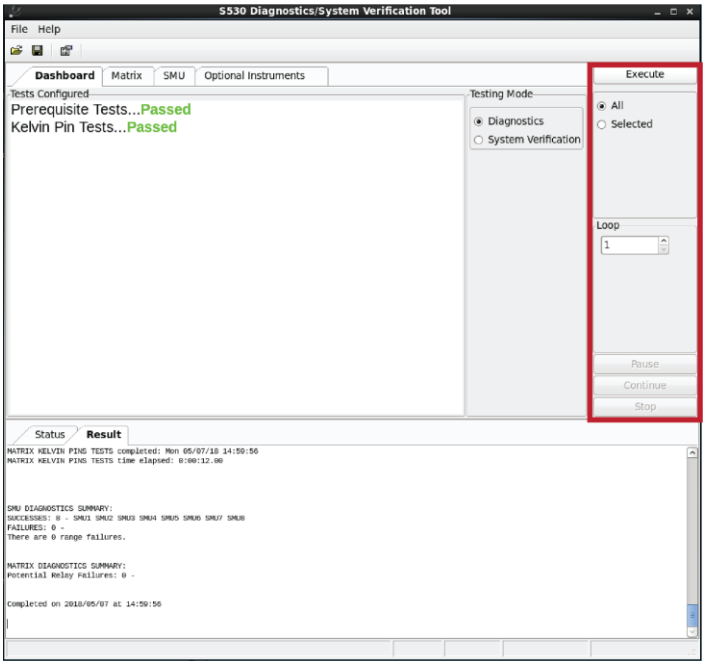
- Dashboard tab (on page 12-7)
- Matrix tab (on page 12-8)
- SMU tab (on page 12-9)
- Optional Instruments tab (on page 12-10)

Use these tabs to select the diagnostic tests you want to execute. The content of each tab is based on the selected instrument type or function.

Control area

The control area is on the right side of the user interface.

Figure 230: Control area



The selections in this area allow you to execute, pause, continue, and stop selected tests. You can also choose a loop count to run one or more tests multiple times.

NOTE

Options available in the control area differ depending on which tab is selected in the selection area (see Selection area (on page 12-4)).

Use the **All** option under the Execute button to select all of the diagnostic tests available on your system. Use the **Selected** option to execute diagnostic tests selected in the Matrix, SMU, and Optional Instruments tabs. The Execute button and the All and Selected options are only available when the Dashboard tab is selected.

You can use the **Loop** number to specify how many times to execute a test (or tests). You can enter the number directly in the text box or use the up and down arrows to adjust the number. The maximum number of loops is 100.

The **Pause** button pauses the diagnostic test that is running. The **Continue** button causes the diagnostic test to continue after a pause event.

NOTE

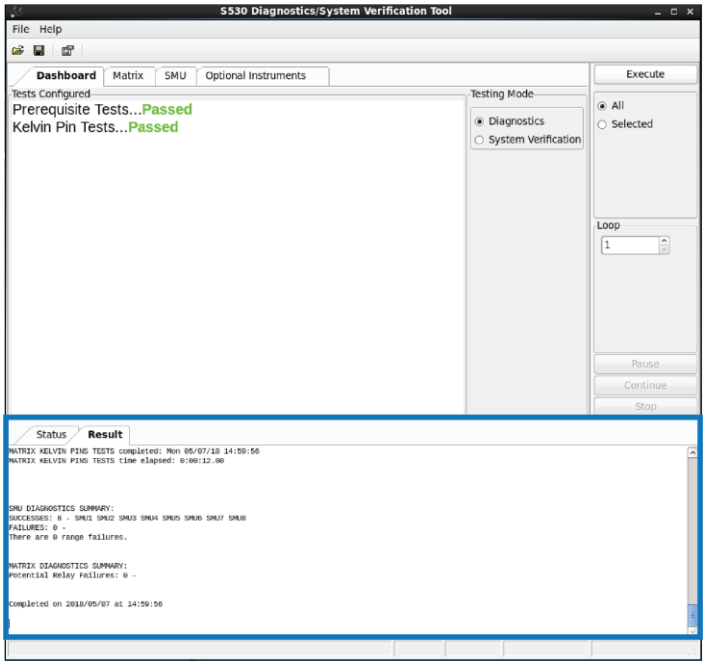
Testing is paused between diagnostic tests, not in the middle of a test. The instrumentation is placed in a safe state at this time.

The **Stop** button allows you to stop the test that is running. Once the testing is stopped, it can only be restarted from the beginning.

Status and result area

The status and result areas are in the lower half of the user interface.

Figure 231: Status and Result area



This area of the user interface contains the Status and Result tabs. These tabs display test information as the diagnostic tests execute. To monitor the testing process, select the **Status** tab to see the status of a test, or select the **Result** tab to view test result information.

You can clear or save the status and result information to a `.txt` file by right-clicking within the each tab. If you choose **Save**, a dialog box is displayed and you must choose where you want to save the contents of the tab.

During execution of a test, the Status and Result tabs provide information pertaining to executed tests. Any time you want to interrupt the automatic scrolling during an active diagnostics session, click the window and scroll. If you want the automatic scrolling to resume, press **Ctrl** and **End** on your keyboard.

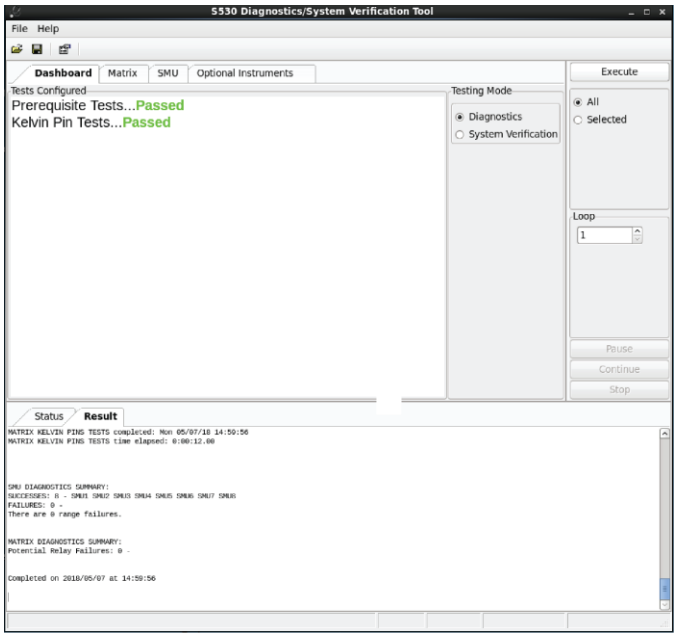
Selection area tabs

The content of the selection area tabs varies depending on the selected instrument type and function. The following topics describe the selection area tabs.

Dashboard tab

The Dashboard tab, shown in the following figure, provides an overview of the test selection and configuration.

Figure 232: Dashboard tab



The Tests Configured area of the tab is a read-only list of the diagnostic tests selected. If you select the **All** button on the right side of the interface (the control area), all of the diagnostics tests are displayed. Choose the **Selected** option to show specific tests.

The Tests Configured list updates during test execution to provide a high-level summary of the diagnostics testing state.

You can choose between executing only diagnostic tests or system verification tests in the Testing Mode area of the tab. The system verification tests verify that the instrumentation is operating within specifications. They take more time to execute than the diagnostic tests.

NOTE

To run system verification tests, all system diagnostic tests must first successfully pass.

Matrix tab

You can select and execute the following tests in the Matrix tab (shown in the following figure):

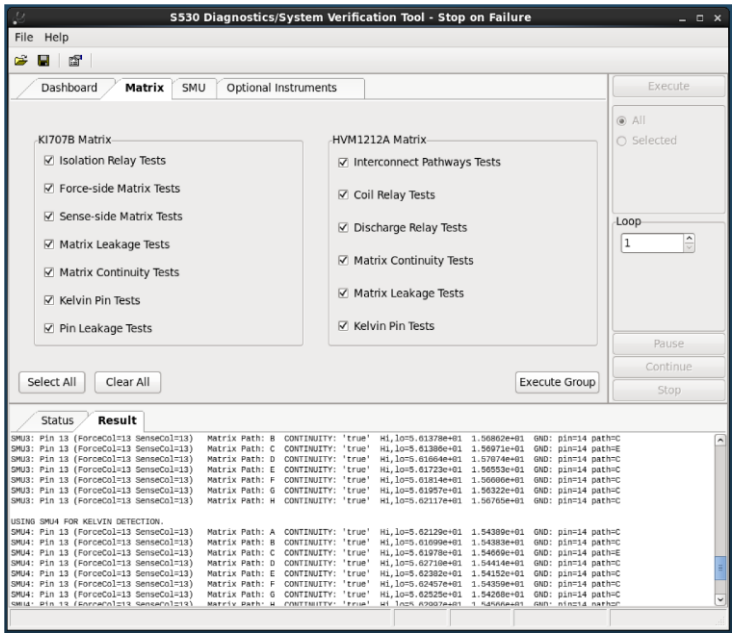
707B matrix

- Isolation Relay Tests
- Force-side Matrix Tests
- Matrix Leakage Tests
- Matrix Continuity Tests
- Kelvin Pin Tests
- Pin Leakage Tests

HVM1212A matrix

- Interconnect Pathways Tests
- Coil Relay Tests
- Discharge Relay Tests
- Matrix Continuity Tests
- Matrix Leakage Tests
- Kelvin Pin Tests

Figure 233: Matrix tab



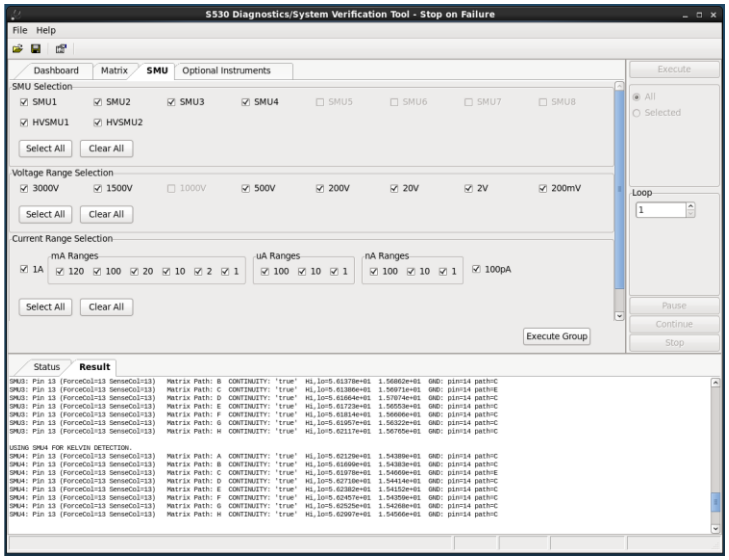
You can choose one or more tests individually by clicking the **Selected** option on the right side of the interface (the control area, shown in the following figure) and selecting the tests you want to run in the selection area of the tab. Select all of the tests by choosing **All** in the control area. The **Execute Group** button enables execution of the selected tests.

Figure 234: The control area of the user interface

SMU tab

You can select and execute the SMU diagnostics tests in the SMU tab (shown in the following figure).

Figure 235: SMU tab



You can select all SMUs or specific SMUs for testing. You can also select all SMU ranges, specific voltage ranges, or specific current ranges to test.

NOTE

When you select voltage and current ranges, those ranges are tested on all of the selected SMUs.

The **Execute Group** button enables the execution of the selected tests.

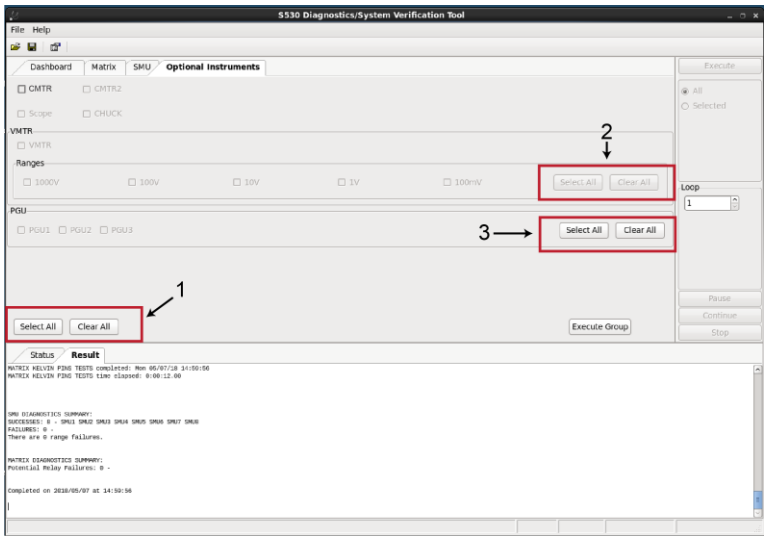
When the SMU tab is selected, the **Execute** button and the **All** and **Selected** options are not available. **Pause**, **Continue**, or **Stop** test sequence controls are still available.

Optional Instruments tab

The Optional Instruments tab, shown in the following figure, allows you to select and execute only the diagnostics tests available for optional instruments:

- Capacitance meter (CMTR)
- Oscilloscope (Scope)
- Digital multimeter (VMTR)
- Pulse generator (PGU)
- Prober chuck (CHUCK)

Figure 236: Optional Instruments tab



Choose the **Select All** or **Clear All** options at the bottom of the tab (see number 1 in the previous figure) to select or clear all selected instruments or select all available options.

Choose the **Select All** or **Clear All** buttons in the VMTR part of the tab to select or clear all of the available voltage ranges for the VMTR (see number 2 in the previous figure).

Use the **Select All** or **Clear All** buttons to select all of the available PGUs (up to three) or clear all selected PGUs (see number 3 in the previous figure).

The **Execute Group** button enables execution of the selected tests.

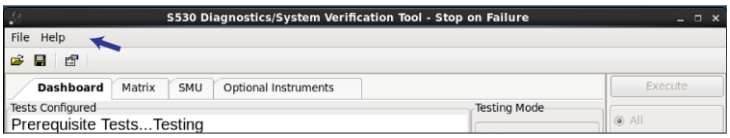
When the Optional Instruments tab is selected, the **Execute** button and the **All** and **Selected** options are not available. **Pause**, **Continue**, or **Stop** test sequence controls are still available.

Menu and toolbar selections

Menu bar

You can load and save diagnostic tests in the user interface using the **File** menu.

Figure 237: Menu



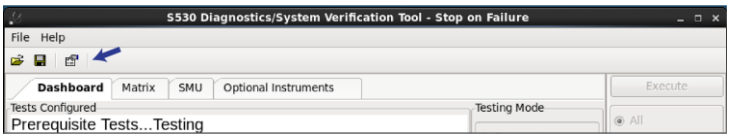
The File and Help menus are summarized in the following table.

Menu selection	Description
Load Test Selections	Preload a previously saved test configuration.
Save Test Selections	Save a test selection for later use.
Exit	Exit the program.
Help	Online help for the user interface.

You can also specify the test selection file from the command line with the `-c` and `-n` switches to execute the selected tests.

Toolbar

Figure 238: Toolbar



The toolbar contains the following items:

- Open Document
- Save Document
- Options:
  - **Stop on Failure:** Diagnostics testing stops if a failure occurs. The instrument and matrix connections remain active and connected so that you can troubleshoot the failure.
  - **Pause on Failure:** Diagnostics testing pauses if a failure occurs. The instrument and matrix connections remain active and connected so that you can troubleshoot the failure. You can press the **Continue** button to allow testing to proceed after a failure.
  - **Continue on Failure:** Diagnostics testing continues testing when a failure occurs.

Command-line arguments

The following command-line arguments are supported by the diagnostics software user interface:

`-c testSelectionFileName`	Load the test selection file.
`-o resultsFileName`	Set the output file name to `resultsFileName`.
`-s`	Enable stop on failure mode.
`-p`	Enable pause on failure mode.
`-n`	Test without a GUI; all tests are executed unless the `-c` option is used.
`-k`	Execute Kelvin tests only, without GUI.

If the `-c` and `-n` switches are used together, the diagnostics system immediately executes the tests. You can use these switches to automate and integrate the diagnostics program into your workflow.

Configuration

The instrument controller configuration file (`$KIHOME/IC/icconfig_<QMO>.ini`) contains a list of instruments to test. This list is in the `DIAG` item of the `[INSTRUMENTATION]` section of the `.ini` file. The S540 diagnostic program reads the instruments from this list and performs tests only on these instruments.

NOTE

The `icconfig_<QMO>.ini` file was configured specifically for your system at the factory. Do not edit the file manually; if changes must be made, contact your Keithley field service engineer (FSE).

NOTE

You must install the blank probe card in the system before running diagnostics. There must be at least two pins configured and connected from the matrix to the probe card through the probe card adapter (PCA). For high-voltage systems, there must be at least three pins configured and connected from the matrix to the probe card through the PCA.

⚠ WARNING

The 9139A probe card may not be used in testing that exceeds 200 V. Testing in excess of 200 V may damage test equipment, or cause injury or death due to electrical shock.

Troubleshooting diagnostics software startup

If you are having trouble starting the S540 diagnostics software, review the following topics for possible solutions to the problem.

If you do not find a solution to your problem here, contact your Keithley field service engineer (FSE).

Project environment settings

To run the diagnostics software, the S540 project environment must be set to the default setting.

If you have your system set up to use a different project environment, the diagnostics software will return an error similar to the following:

```
[kthmgr@localhost usrlib]$ diags
<program name unknown>: error while loading shared libraries: libS540_3kV_Diags.so:
   cannot open shared object file: No such file or directory
the result thread exits due to missing diagnostics server
```

When you get this error, you must set the project environment to the default setting if you want to run the diagnostics software.

To change the project environment to the default setting:

1. At the command prompt, type `select_project`. The name of your present project is returned, as shown below:

```
[kthmgr@localhost ~]$ select_project
Your choices are:
Project 1 is current.env
Project 2 is Keithley_Orig.env
Project 3 is Custom.env
-------------------------
Your current project -->> Custom.env
-------------------------
Enter your selection...
 or   0 to cancel
     111 to load KI Original
```

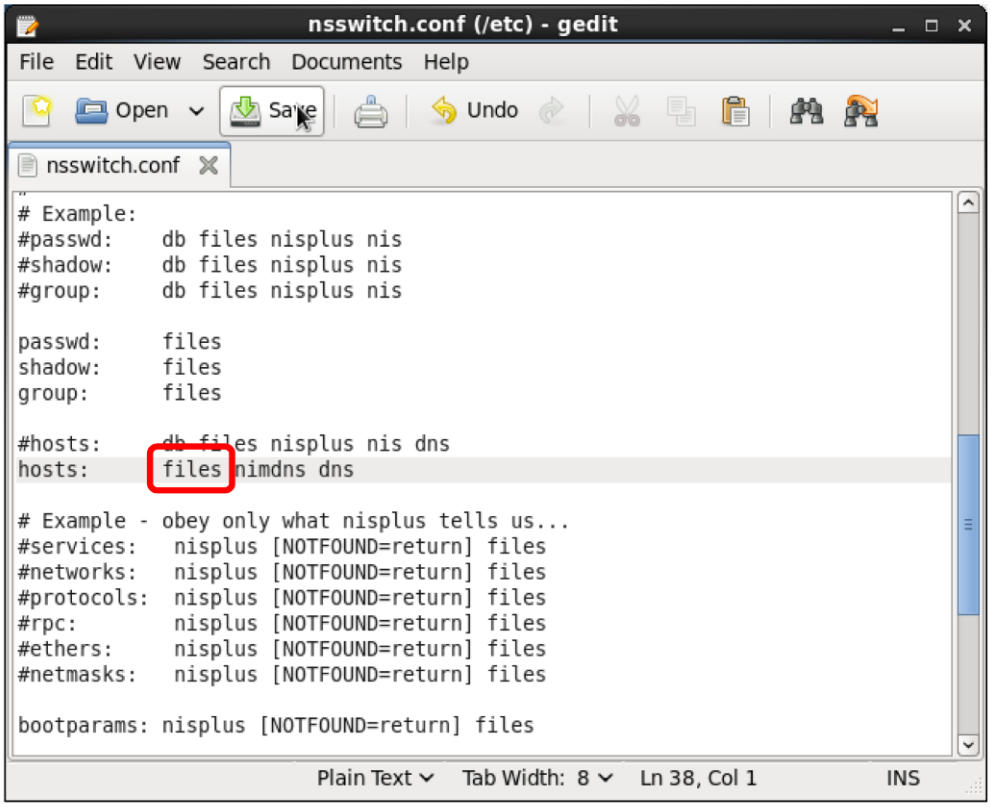
2. At the prompt, enter `111` to load the KI Original project environment (or enter `0` to cancel the `select_project` function).

Hosts entries in nsswitch.conf file

If you are having problems starting the S540 diagnostics program, the problem may be the order in which hosts are listed in the `/etc/nsswitch.conf` file.

To check this, open the `nsswitch.conf` file and make sure that the `files` host is listed first on the `hosts:` line, as shown in the following figure.

Figure 239: Correct order for hosts entries in nsswitch.conf



Test sequence

The following list is an outline of the diagnostic testing sequence. Each of these steps is described in detail in the rest of this section.

System information (on page 12-16)

- Details about system instruments, matrix cards, software, and relay usage.

Prerequisite tests (on page 12-16)

- **SMU 2-wire prerequisite tests:** Minimum of two source-measure unit (SMU) channels
- **SMU 4-wire prerequisite tests:** Minimum of two passing SMUs (707B matrix only)
- **Kelvin pin test:** Minimum of two Kelvin pins

Diagnostic tests (on page 12-21)

- **Kelvin pin tests:** All pins must have good Kelvin connections
- **Matrix tests:**
  - Isolation relay tests
  - Force-side relay tests
  - Matrix leakage tests
  - Matrix continuity tests
  - Pin leakage tests
  - Pin guard tests
  - Pin settling tests
  - High-voltage matrix interconnect pathway tests
  - High-voltage matrix coil relay tests
  - High-voltage matrix discharge relay tests

SMU tests (on page 12-38)

- SMU voltage and current range tests

Tests for other instruments (on page 12-40)

- Capacitance meter (CMTR) tests
- Oscilloscope (Scope) tests
- Pulse (PGU) tests
- Digital multimeter (voltmeter (VMTR)) tests

System information

Before prerequisite tests begin, the diagnostics software queries the system to identify all of the instruments and matrix cards in the S540 system.

Information shown in the diagnostics log includes:

- **System:** Name of the S540 system (for example, S5400q3333)
- **Model:** System type (for example, S540)
- **KTE Version:** Version of the Keithley Test Environment (KTE) software installed on the system
- **Date:** Date of the diagnostics test
- **Time:** Time of the diagnostics test
- **Instrument Information:** Name (for example, SMU1 or MTRX1), manufacturer name, model number, serial number, firmware revision, and calibration date for each instrument in the S540 system
- **Matrix Card Information:** Slot number, model number, card description, version, and serial number for each matrix card in the S540 system (this applies only to installed cards in the 707B Switch Matrix Mainframe)
- **S4200A-SCS System Information:** Serial number and Keithley Test Environment Interactive (KTEI) software version
- **S4200A-SCS Instrument Information:** List of instruments and cards installed in the S4200A system, including the model number, serial number, revision, and calibration date for each (as applicable)
- **HVM Relay Usage:** The number of times each relay has opened and closed; the relays in the HVM1212A have a rated lifetime of 100-million cycles for each contact

Prerequisite tests

The prerequisite tests determine if the instruments in the system are working properly, if a minimum number of source-measure units (SMUs) exist in the system, and if matrices in the system are functioning. These preliminary tests are very simple tests that are not intended to verify full functionality.

The prerequisite tests must successfully complete before the entire diagnostics test suite can execute.

NOTE

You must install the blank probe card in the system before running diagnostics. There must be at least two pins configured and connected from the matrix to the probe card through the probe card adapter (PCA). For high-voltage systems, there must be at least three pins configured and connected from the matrix to the probe card through the PCA.

⚠ WARNING

Hazardous voltages may be present on the probe card adapter, even after you disengage the interlock. Cables can retain charges after the interlock is disengaged, exposing you to live voltages that, if contacted, may cause personal injury or death.

Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

SMU 2-wire prerequisite tests

The purpose of this test is to make sure connections between SMUs and matrices in the system are valid. One source-measure unit (SMU) forces voltage or current into another SMU, for each matrix pathway. Separate tests are done for Model 7531 switch cards in the Model 707B switch matrix mainframe and for the Model HVM1212A high-voltage matrix.

The SMUs are placed in 2-wire mode for the test.

SMU 2-wire prerequisite test for SMUs with 707B matrix

The following sequence is completed for each source-measure unit (SMU) in your system.

SMU 2-wire prerequisite test sequence (SMUs and 707B matrix):

1. Forces voltage on the forcing SMU and measures voltage with the measuring SMU.
   Forces and measures the following voltages: ±1.4 V, ±12.3 V, and ±32.7 V.
2. Repeats step 1, except it forces and measures current instead of voltage.
   Sets a limit of 5.0 V and forces ±1.4 mA and ±1.4 µA.
3. Swaps the forcing and measuring SMUs and repeats the test.

Results: The force and measure test for the SMU combination must pass for all matrix pathways before a SMU is considered functional.

SMU 2-wire prerequisite test for 2657A SMUs with HVM1212A matrix

If your S540 system has 2657A source-measure units (SMUs) and an HVM1212A 3-kV matrix, the following sequence is completed for each 2657A.

SMU 2-wire prerequisite test sequence (2657A SMUs and HVM1212A matrix):

1. High-voltage SMU1 (HVSMU1) forces 1 mA through pathway A on the HVM1212A matrix to HVSMU2.
2. HVSMU1 measures current to be sure it is actually forcing 1 mA.
3. HVSMU2 measures current to be sure the signal is getting through pathway A of the HVM1212A matrix to HVSMU2.
4. The diagnostics software returns `SUCCESS` or `FAILURE` for the pathway A connection.
5. Steps 1 through 4 are repeated for each of the remaining pathways (B, C, D, E, F, G, H)
6. The entire sequence is repeated, but with HVSMU2 as the forcing SMU and HVSMU1 as the measuring SMU.

Results: If the diagnostics log shows `FAILURE` for any of the force-measure tests, there is a problem with that specific connection. Reasons for test failure:

- The forcing SMU is not forcing current.
- The measuring SMU is not measuring current.
- The measured current value is not within five percent of the programmed 1 mA value.

SMU 4-wire prerequisite test

NOTE

The diagnostics software does not run the 4-wire test on 2657A source-measure units (SMUs) — only on 2636B SMUs.

The source-measure unit (SMU) 4-wire test follows the same sequence as the 2-wire test, except that the SMUs are placed into 4-wire mode.

Kelvin pin tests

This prerequisite test scans all pins with the two functional source-measure units (SMUs) identified in the SMU 2-wire prerequisite test. It verifies that all pins have valid Kelvin connections.

If any of the pins fail, the test fails and further testing cannot continue.

2657A Kelvin pin test

The 2657A source-measure units (SMUs) also have a built-in contact check feature that determines if a Kelvin connection is present.

The diagnostics software runs a test similar to the 2636A/B test on the 2657A 3-kV SMUs in your system, as described below.

2657A Kelvin pin test sequence:

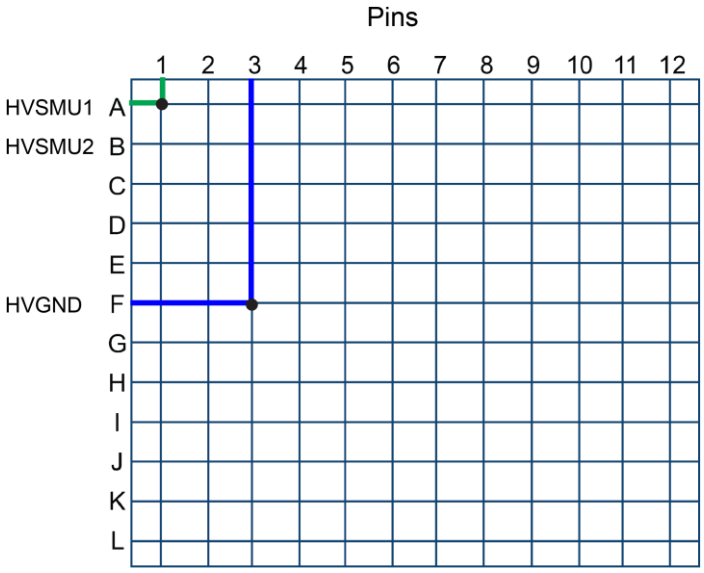
For the 2657A 3000 V source-measure units (SMUs), the diagnostics software does the following:

1. Connects HVSMU1 to row A and HVGND to pin 3.
2. Closes the A01 and F03 relays.
3. Executes contact check on HVSMU1
4. Opens the relays.
5. Repeats steps 1 through 4, incrementing the pin tested each time. When the pin that HVGND is connected to (pin 3) is checked, the HVGND connection is moved to another pin that is not connected to HVSMU1.

Results: All pins must have valid Kelvin connections to pass this test and continue with full diagnostic testing.

The following figure shows Kelvin pin prerequisite testing on pin 1.

Figure 240: HVSMU1, pin 1, and ground connected



Kelvin pin prerequisite test

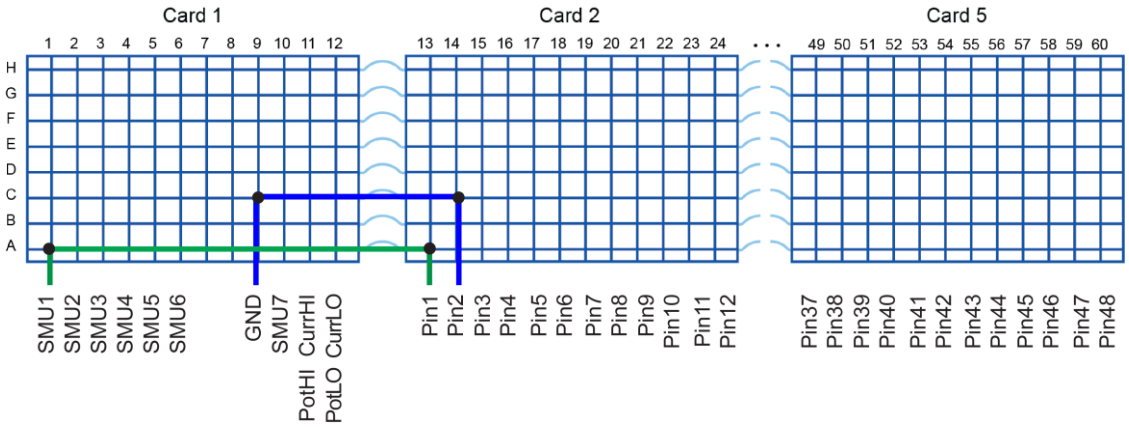
The source-measure units (SMUs) in the S540 have a built-in contact check feature that determines if a Kelvin connection is present.

Kelvin pin prerequisite test sequence:

1. The diagnostics software connects SMU1 with the first pin using matrix pathway A.
2. If a valid Kelvin connection exists, the same SMU-pin pair is connected again using the next matrix pathway.
3. This loop is repeated for each pathway. If all pathways have a valid Kelvin connection, the SMU-pin pair is good.
4. The diagnostics software repeats steps 1 through 3 with SMU1 connected to each remaining pin.
5. The diagnostics software then tests the remaining SMUs in the system using only pin 1 with all of the pathways (this is because the other pin combinations were previously tested with the first SMU).

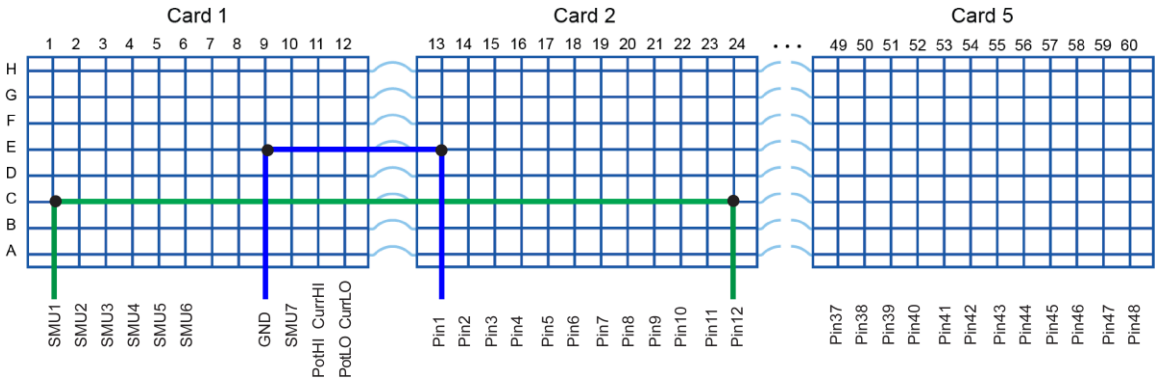
Results: If any of the configured pins do not have a valid Kelvin connection, the test fails.

Figure 241: SMU1 and pin 1 connected



The following figure shows SMU1 and pin12 connected during the Kelvin-pin detection. The blue pathway shows pin1 connected to GND to complete the LO-sense LO portion of the Kelvin connection. In this example, matrix pathway C is tested for Kelvin operation and matrix pathway E is used for the GND connection.

Figure 242: SMU1 and pin 12 connected



Diagnostic tests

Once prerequisite testing has completed and all tests have passed, diagnostic testing can begin. The following topics describe the diagnostic tests.

NOTE

The diagnostics software overrides any `MAX_VOLTAGE` settings you may have specified in the `icconfig_<QMO>.ini` file.

Installed probe card

Before the diagnostic tests begin, the diagnostic software determines the type of probe card that is installed in the system. You can run diagnostics with a blank probe card in your system, but if you want to do system verification, you must install the shorted probe card that came with your system.

⚠ WARNING

Hazardous voltages may be present on an installed probe card even after the output is disconnected, that if contacted, may cause personal injury or death.

Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

Kelvin pin tests (4-wire systems only)

This diagnostic test is the same as the prerequisite Kelvin pin tests (on page 12-18).

You can run this Kelvin pin-connection test separately to validate connections from the system to the probe card. This is useful after changing the probe card or recabling. See Matrix tab (on page 12-8) for information about how to configure the diagnostics software to run this test separately.

Matrix tests

After prerequisite tests complete successfully, matrix diagnostic testing can be done. The diagnostics software runs the following matrix tests:

- Isolation relay test (on page 12-24)
- Force-side relay tests (on page 12-27)
- Matrix leakage test (on page 12-29)
- Matrix continuity test (on page 12-32)
- Kelvin pin test (4-wire systems only) (on page 12-22)
- Pin leakage tests (on page 12-34)

These tests are run in the order shown here in comprehensive diagnostics testing. You can also choose any one of these tests (or a combination of them) and run them in a separate test run.

Matrix arrangement

For S540 systems, the matrix arrangement differs depending on which system configuration you have:

- 3-kV system with a low-current 707B matrix
- 3-kV system with no low-current matrix

The following figures show the differences in the matrix arrangement and card numbering.

Figure 243: S540 3-kV high-voltage, low-current matrix arrangement

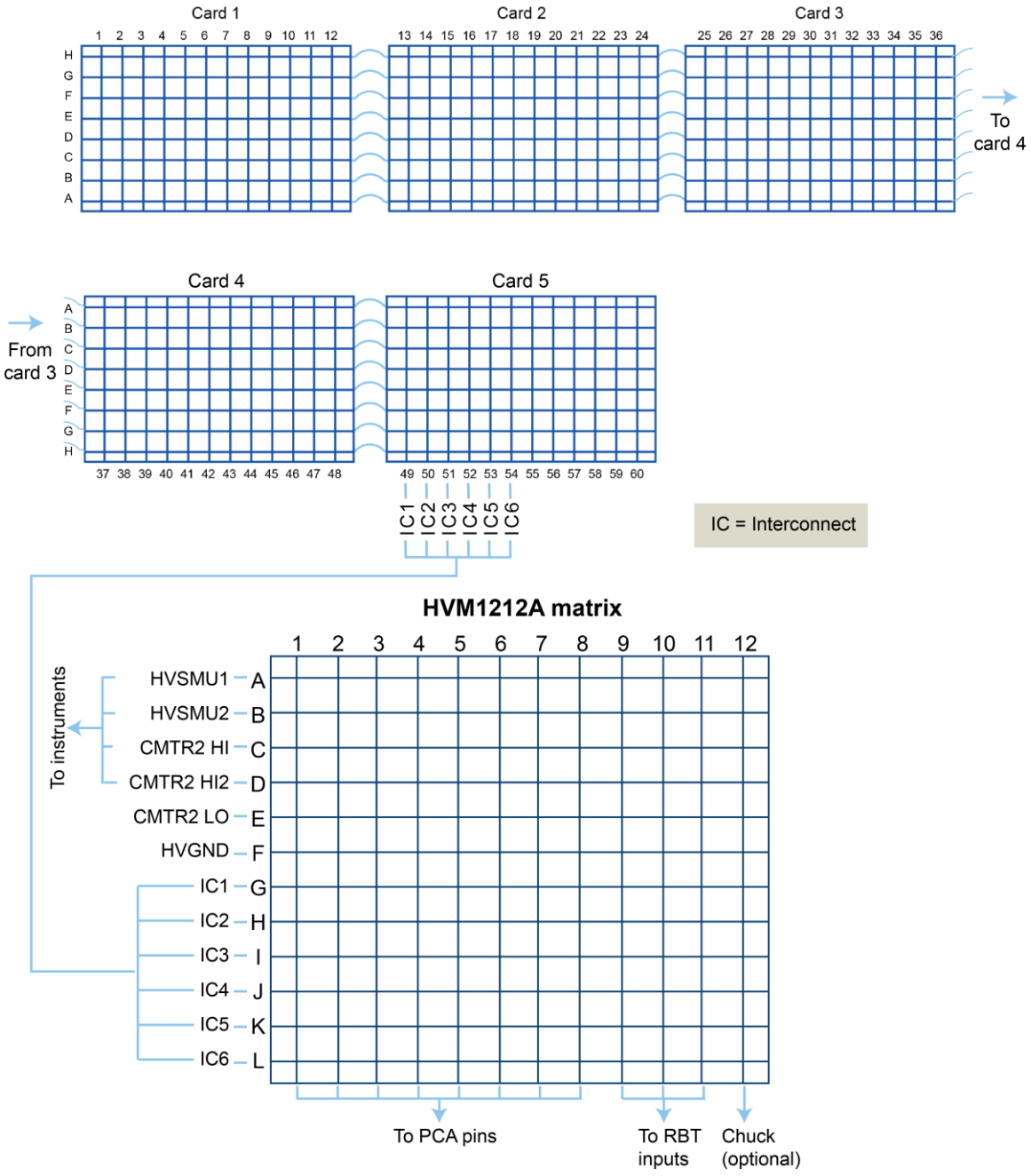
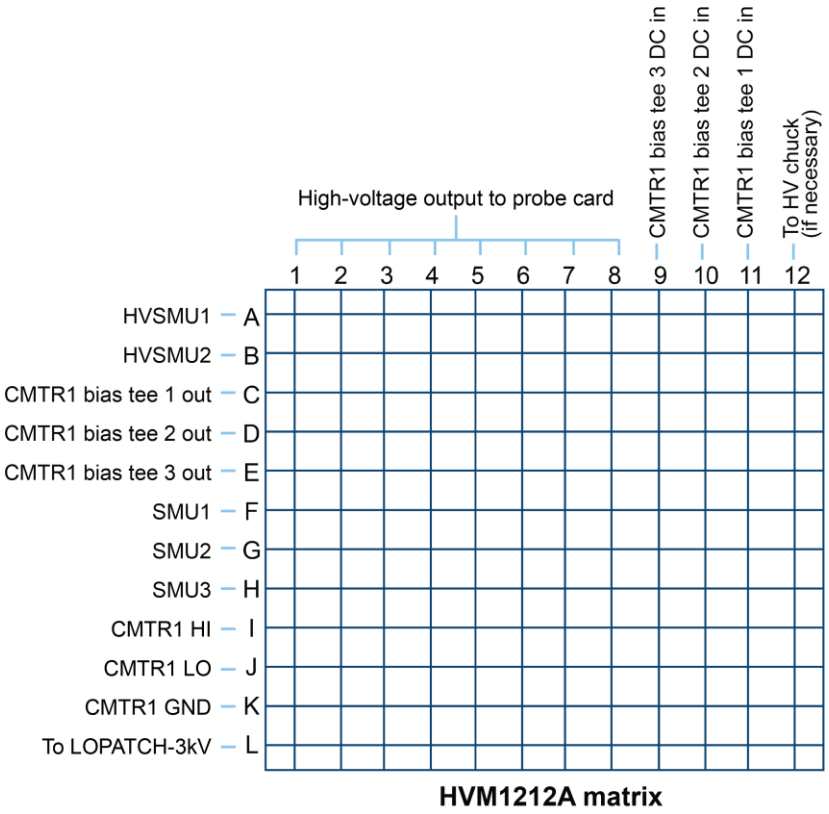


Figure 244: S540 3-kV high-voltage only matrix arrangement



Isolation relay tests

The 7531 low-current relay cards in the S540 KTE systems contain isolation relays for row connections. The 707B mainframe cannot directly control the isolation relays. The isolation relay tests validate that these relays function correctly.

Requirements: One working source-measure unit (SMU) and Kelvin connections on all pins.

NOTE

During testing, the SMU is placed in 2-wire mode.

Force-side row isolation relays

The test first validates that the working source-measure unit (SMU) (SMU1 in this case) can force current into ground (GND) for all eight pathways on card 1.

Force-side row isolation relays test sequence:

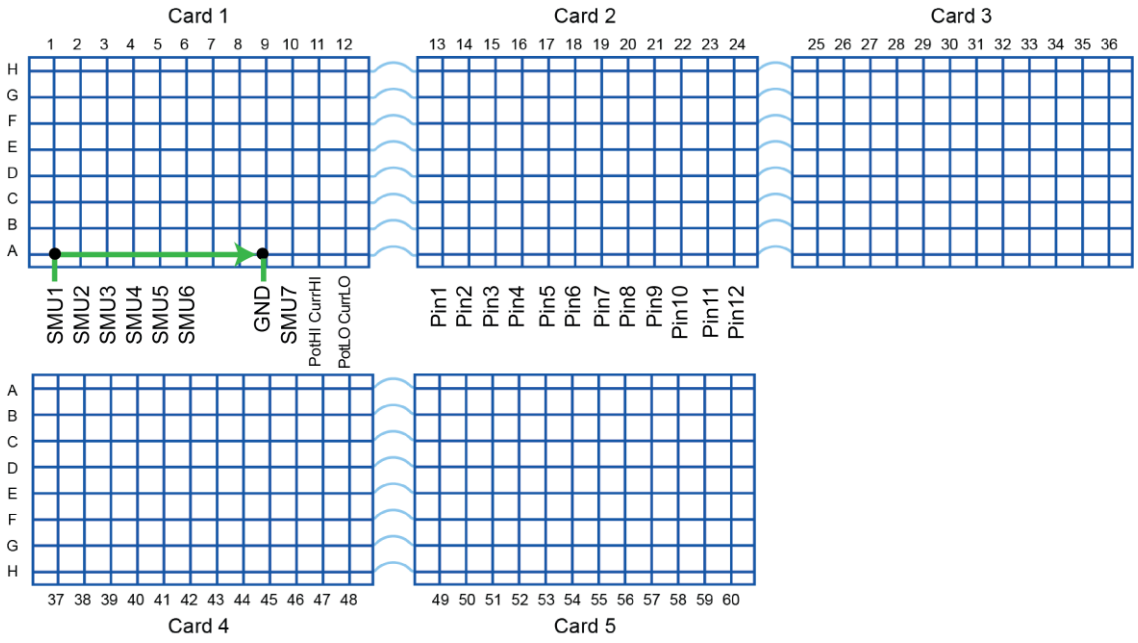
1. SMU1 is configured with a voltage limit of 5 V and forces 1 mA of current into ground.
2. Current flowing out of SMU1 is measured and compared with 1 mA.
3. If the current measurement is correct, step 1 and step 2 are repeated on pathways C to D, E to G, and F to H.
4. Step 1 and step 2 are repeated until all eight pathways are verified.
5. Once all card 1 pathways pass the test, the diagnostics software connects card 1 to card 2 connections followed by card 1 to card 3 connections to verify the row isolation relays. Because card 1 cannot be tested directly, both card combinations must be tested to determine if card 1, card 2, or card 3 has any failures.

NOTE

The relay is tested two times in both open and closed states to increase the chances of detecting a relay that may be intermittently stuck in one state.

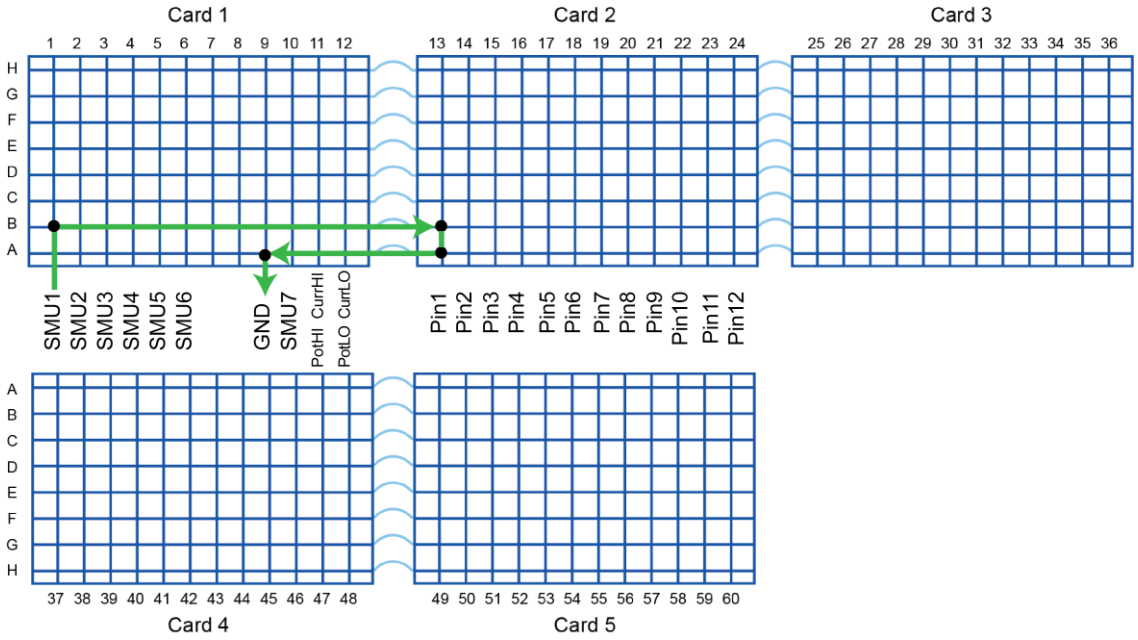
The following figure shows the matrix connection for testing pathway A.

Figure 245: Validate all pathways for card 1



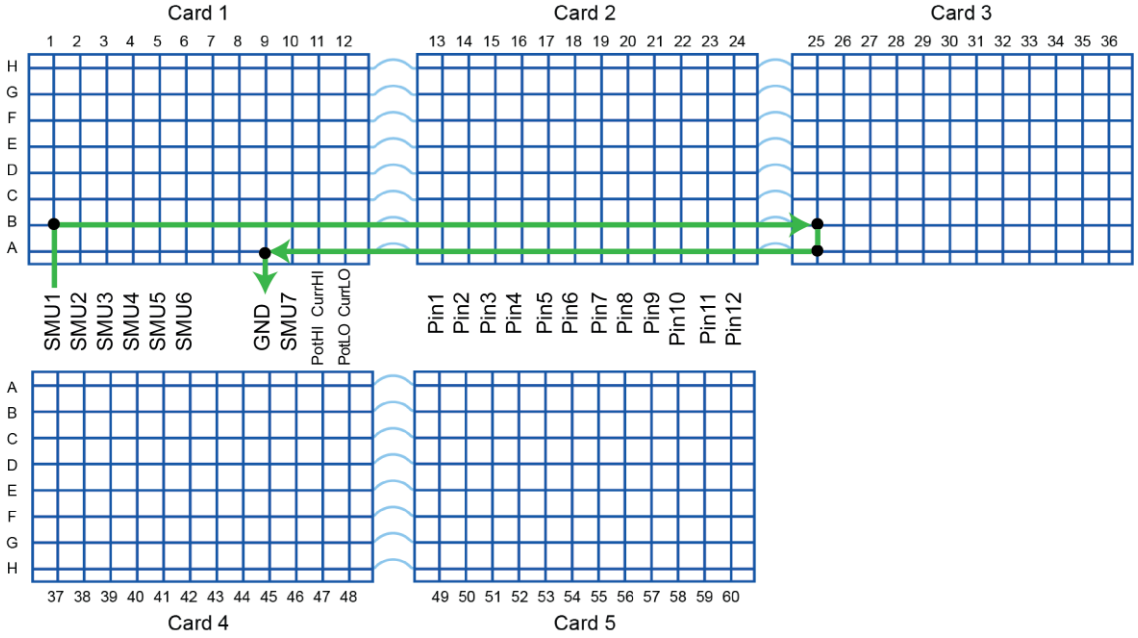
The following figure shows the matrix connections to verify pathways A and B.

Figure 246: Check card 1 and card 2 isolation relays



The following figure shows the connections to check card 1 and card 3 isolation relays.

Figure 247: Check card 1 and card 3 isolation relays



Results: If all of the above tests pass and fail at the appropriate times, the row isolation relays for pathway A to B between card 1 and card 2 are operational. If the tests fail, the row isolation relays on either card 1 or card 2 are bad (the test does not identify which card is bad).

Once testing has completed for card 3, diagnostics determines whether row isolators of card 1 are working. If both card 2 and card 3 tests are good, then card 1 is also good.

If the card 1 to card 2 tests fail and the card 1 to card 3 tests are good, card 2 should be replaced.

If the card 1 to card 2 tests are good and the card 1 to card 3 tests fail, card 3 should be replaced.

If the card 1 to card 2 tests fail and the card 1 to card 3 tests fail, card 1 should be replaced.

Test details

During the test, current is forced and measured with SMU1. If current flows, the test concludes that the row isolators for card 1 and card 2 are closing. The test must also verify that these isolation relays open correctly. The complete test sequence is shown in pseudo code below.

```
close relay 2A01 and 2B01
/*Measure 1mA */
forcei/measi
devclr()
open relay 2A01
forcei/measi  /* should fail current flow */
devclr()
close relay 2A01
forcei/measi  /* should pass again */
devclr()
open relay 2B01
forcei/measi  /* should fail current flow */
devclr()
close relay 2B01
forcei/measi   /* should pass again */
devclr()
open relays 2A01 and 2B01 since this test is complete.
devint()
```

Force-side relay tests

This test verifies that each relay in the system operates correctly.

For the 707B low-current matrix, the diagnostics software uses the following procedure during the test:

- The eight rows of the matrix are processed as four groups of two: A to B, C to D, E to G, and F to H.
- Two relays from a column are tested at one time; one relay from each row of the present group.

NOTE

Before testing the relays, the routine verifies that two source-measure units (SMUs) operate correctly and connect to all matrix pathways.

- The SMU is placed in 2-wire test mode.
- The test connects a circuit from a forcing SMU to ground (GND).
- The test forces 1 mA and measures voltage and current using the SMU (voltage limit set to 5 V).

707B force-side relay test

Two relays from a column are tested at one time, with an open-test or close-test approach (similar to the force-side row isolation tests).

Force-side relay test sequence:

1. Closes both relay 1 and relay 2 and measures. Current should flow.
2. Opens relay 1 and measures. Current should not flow.
3. Closes relay 1 and measures again. Current should flow.
4. Opens relay 2 and measures. Current should not flow.
5. Closes relay 2 and measures again. Current should flow.
6. Opens relay 1 and relay 2. The tests for this relay pair are complete.

The following figure shows the two relays that are being tested (in red) with the open and close process described in the steps above.

In the Status and Result logs, each relay closure is described by a set of numbers and letters to show the card, row, and column. For example, 1A01 is card 1, row (or path) A and column 1.

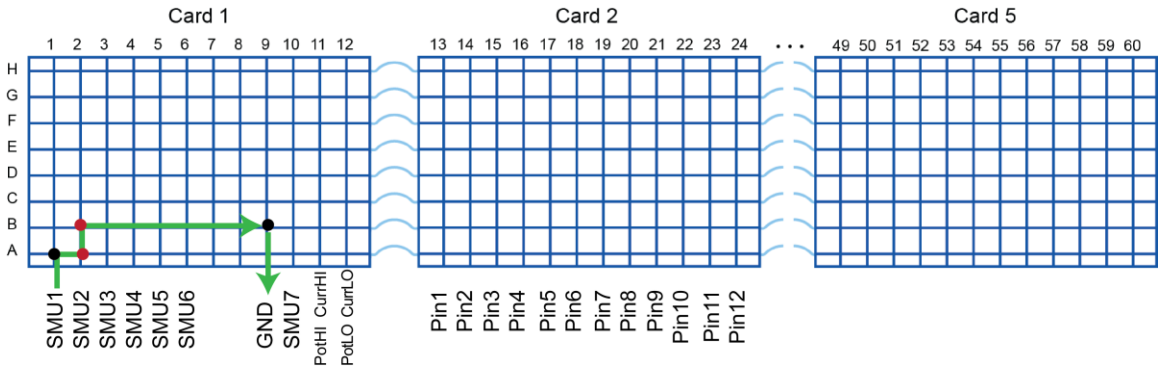
NOTE

The column numbers in the results log are card-based, not system-based.

The following figure shows the connections:

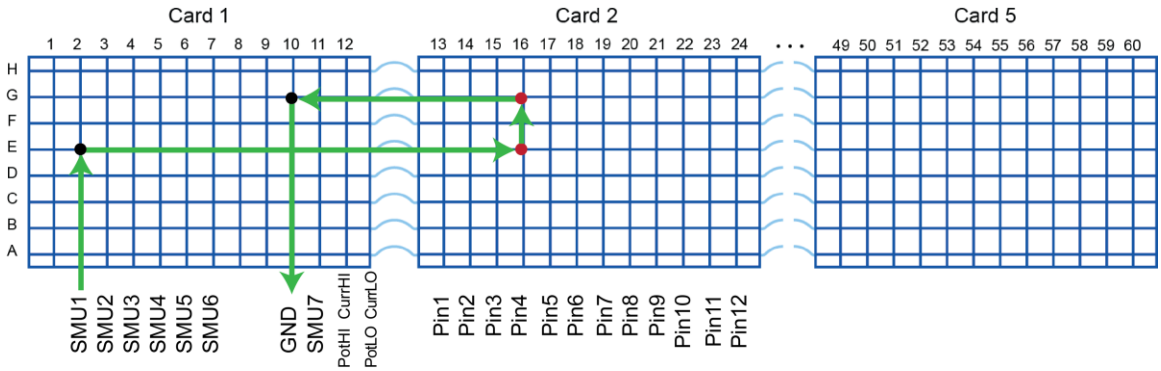
SMU1 1A01,1B09, 1A02,1B02

Figure 248: Testing force-side relays on card 1



The following figure shows a later step in the 2-wire relay test.

Figure 249: Testing force-side relays on card 2



Matrix leakage tests

The matrix leakage test detects if there is a short or high leakage between adjacent matrix pathways. Two source-measure units (SMUs) are used: A forcing SMU and a measuring SMU. The SMUs are connected to the matrix in 4-wire mode.

HVM1212A high-voltage matrix leakage test

For the HVM1212A 3-kV matrix leakage test, diagnostics performs the following sequence.

NOTE

The diagnostics software overrides any `MAX_VOLTAGE` settings you may have specified in the `icconfig_<QMO>.ini` file.

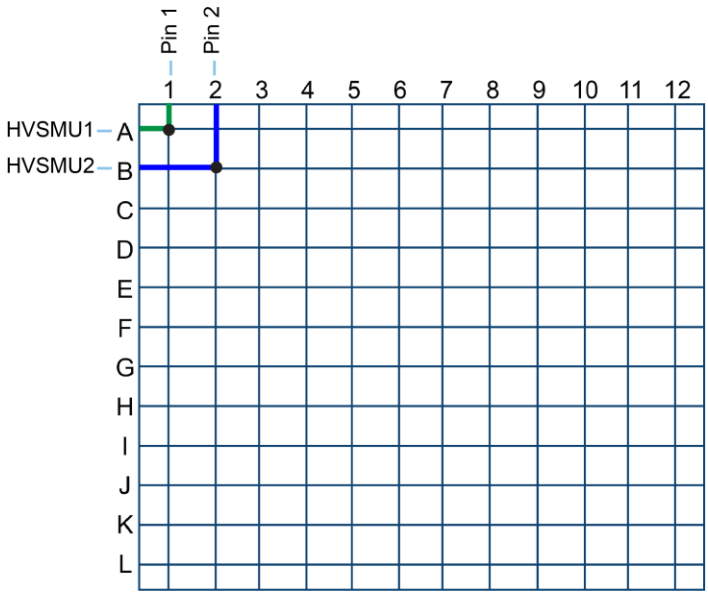
High-voltage matrix leakage test sequence:

1. HVSMU1 forces 200 V on pin 1 and HVSMU2 measures current on pin 2.
2. HVSMU2 forces 200 V on pin 1 and HVSMU1 measures current on pin 2.
3. This sequence is repeated on pin 2 to pin 3, pin 3 to pin 4, pin 4 to pin 5, pin 5 to pin 6, pin 6 to pin 7, pin 7 to pin 8, and pin 8 to pin 1.
4. HVSMU1 forces 3000 V on pin 1 and HVSMU2 measures current on pin 2.
5. HVSMU2 forces 3000V on pin 1 and HVSMU1 measure current on pin 2.
6. This sequence is repeated on pin 2 to pin 3, pin 3 to pin 4, pin 4 to pin 5, pin 5 to pin 6, pin 6 to pin 7, pin 7 to pin 8, and pin 8 to pin 1.

Result: This test compares measured current to a preset limit. If the measured current is within ±10 pA of the limit, the test passes. If the measured current is outside of ±10 pA, the test fails.

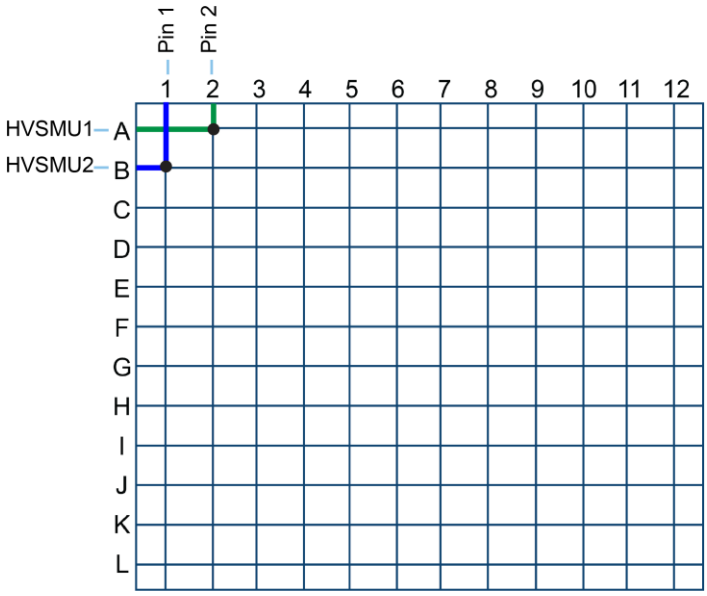
The following figure shows the connections in steps 1 and 4 of the above sequence.

Figure 250: HVM1212A matrix leakage test SMU 1 forcing V, SMU2 measuring I



The following figure shows the connections in steps 2 and 5 of the above sequence.

Figure 251: HVM1212A matrix leakage test SMU 2 forcing V, SMU1 measuring I



707B matrix leakage test

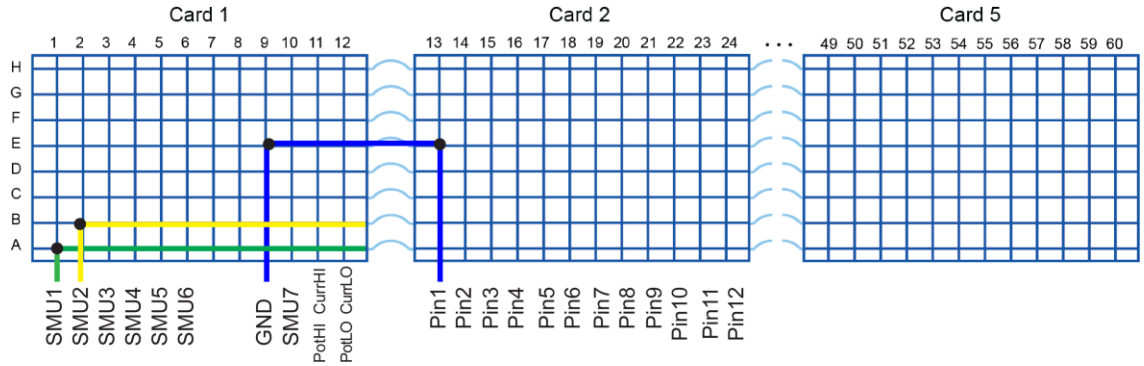
The following sequence is completed for the 707B matrix.

Matrix leakage test sequence:

1. The forcing SMU forces 180 V.
2. The measuring SMU measures the current on the adjacent pathway. A measurement higher than the leakage threshold is a failure; leakage limits vary by system type (see the results information in the user interface for specific leakage limits).
3. The force and measure SMUs are swapped and the force and measure sequence is repeated.

The following figure shows the matrix connections for testing 707B pathway A and B with SMU1 and SMU2. The green path shows the forcing SMU and the yellow path shows the measuring SMU. The blue pathway is the ground connection needed for a valid Kelvin 4-wire configuration.

Figure 252: Pathway shorts test for pathway A to B



Matrix continuity tests

This test measures the resistance of the pathways to make sure that relay contacts are working properly.

HVM1212A high-voltage matrix continuity test

For the HVM1212A 3-kV matrix continuity test, diagnostics performs the following sequence.

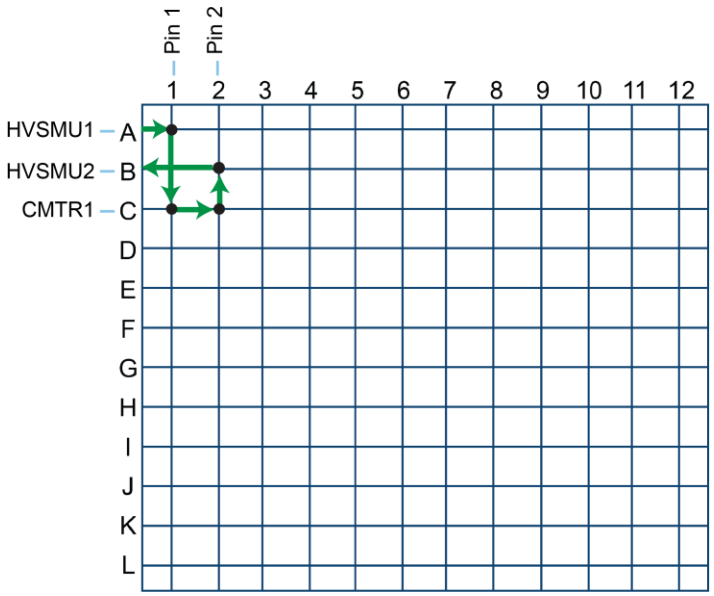
High-voltage matrix continuity test sequence:

1. Connects HVSMU1 and HVSMU2 to each other on a pathway through relays A1, C1, C2, and B2.
2. HVSMU1 forces 1 mA and HVSMU2 forces 0 V.
3. Diagnostics compares the measured outputs to make sure they are correct to within one percent.
4. The process is repeated on each remaining pathway until all pathways are tested. For example, the next pathway is A1, C1, C3, and B3, and the pathway after that is A1, C1, C4, and B4, and so on.

Result: Any pathways with measured outputs that have a greater than one percent difference fail the test.

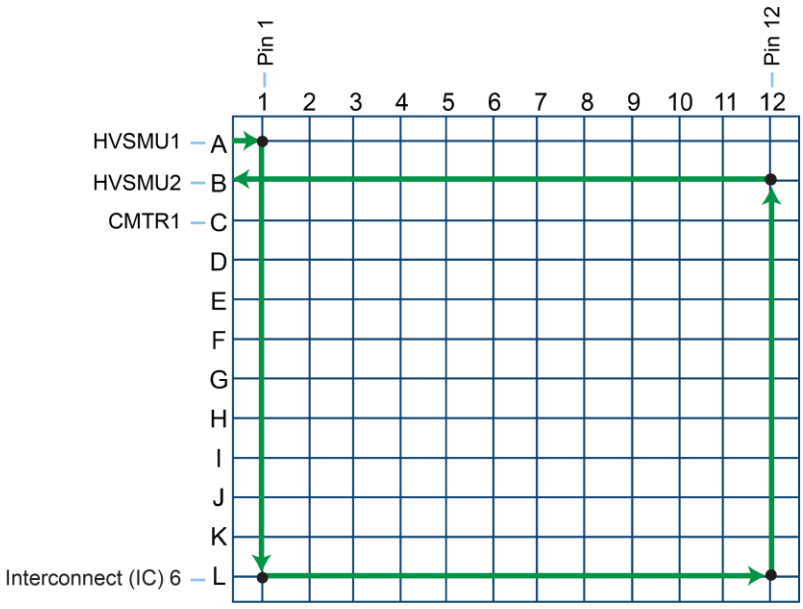
The following figure shows the connections for the first tested pathway in the high-voltage matrix continuity test.

Figure 253: HVM1212A pathway A1, C1, C2, B2 continuity test connections



The following figure shows the connections for the last tested pathway in the high-voltage matrix continuity test.

Figure 254: HVM1212A pathway A1, L1, L12, B12 continuity test connections



707B matrix continuity test

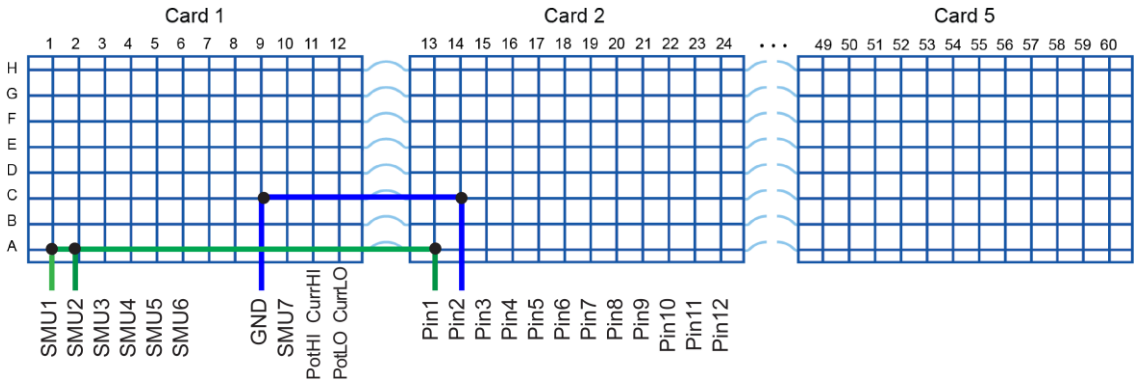
For the 707B low-current matrix, the diagnostics software repeats the following sequence for each pathway.

Matrix continuity test sequence:

1. Connects two source-measure units (SMUs) to each other on a pathway
2. SMU1 forces 1 mA (20 V limit) and SMU2 forces 0 V
3. Measures current and voltage on SMU1 and calculates resistance

Results: Any resistance greater than 2 Ω is a failure.

Figure 255: Pathway A continuity test connection



Pin leakage tests

There are three parts to the pin leakage test:

1. **Pin leakage test:** Check for leakage or shorts between adjacent pathways (for 707B only)
2. **Pin-guard test:** Check for leakage or shorts on the guard signal of the pathways (for 707B with 7531 cards only)
3. **Pin settling test:** Measure overall low-current leakage on a given pin with all other pins tied to ground (for HVM1212A and 707B)

Leakage or shorts between adjacent pathways test

Using the first two good pathways identified in earlier matrix leakage and continuity tests, the routine performs the following test on the adjacent Kelvin pins (shown below in pseudocode).

This test is available only for the 707B switch matrix.

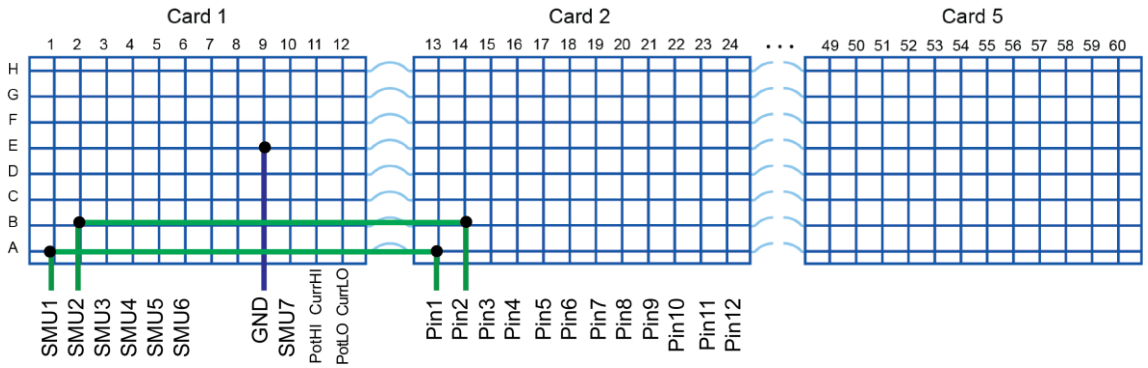
Test sequence for the 707B matrix:

```
conpth(goodPath1, forcingSMU, pinA, 0)
conpth(goodPath2, measSMU, pinB, 0 )
conpth(altPathway, GND, 0 )

forcev(measSMU, 0.0)
forcev(forceSMU, 50.0)
rangei(measSMU, 10e-9)
setmode(KI_SYSTEM, KI_INTGPLC, 5.0)
delay(8000)
intgi(measSMU, leakage)
```

Results: If the leakage is greater than 10 nA, the test fails.

Figure 256: Pin 1-2 leakage test connection



Pin-guard test

The pin-guard test is similar to the pathway shorts test where a quasistatic, SMU-based capacitance measurement is used to test for guard-to-ground shorts. The maximum capacitance value is 150 pF.

This test is available only for the 707B switch matrix with 7531 switch cards.

Pin settling test

The third part of the pin leakage test is the pin settling test. This test is available on both the 707B and HVM1212A matrices

707B pin settling test

707B pin settling test sequence:

1. For each low-voltage pin defined in the system, the test connects a source-measure unit (SMU) and connects all other pins to ground.
2. The test then forces 100 V, delays 1 second, and measures current with the SMU.

Result: A result of more than 100 pA on any pin is a failure.

This test works according to the following pseudocode:

```
conpin(SMU1, test_pin, 0)
conpin(GND, all_other_pins, 0)

limit(SMU, 1e-3)
lorangei(SMU, 1e-10)
rangei(SMU1, 10e-9)
setauto(SMU1)
forcev(SMU, 100)
delay(1000)
intgi(SMU1, settling_leakage)
```

HVM1212A pin settling test

NOTE

The diagnostics software overrides any `MAX_VOLTAGE` settings you may have specified in the `icconfig_<QMO>.ini` file.

High-voltage SMU and digital multimeter (VMTR) pin settling test sequence:

1. For each pin defined in the system, the test connects a high-voltage source-measure unit (HVSMU) and connects all other pins to ground.
2. The test then forces 3000 V, delays 1 second, and measures current with the HVSMU.

Result: A result of more than 4.5 nA after one second on any pin is a failure.

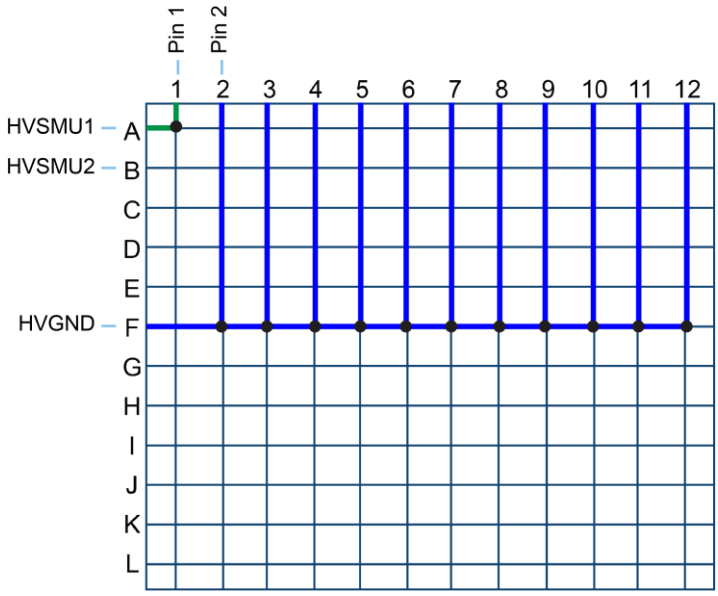
This test works according to the following pseudocode:

```
conpin(HVSMUA, pinA, 0);
conpin(HVSMUB, pinB, 0);

forcev(HVSMUA, rangeV);
forcev(HVSMUB, 0.0);
delay(2000);
intgi(HVSMUB, result);
```

The following figure shows HVSMU1 connected to pin 1 and all other pins connected to ground.

Figure 257: HVM1212A pin settling test



HVM1212A matrix interconnect pathway tests

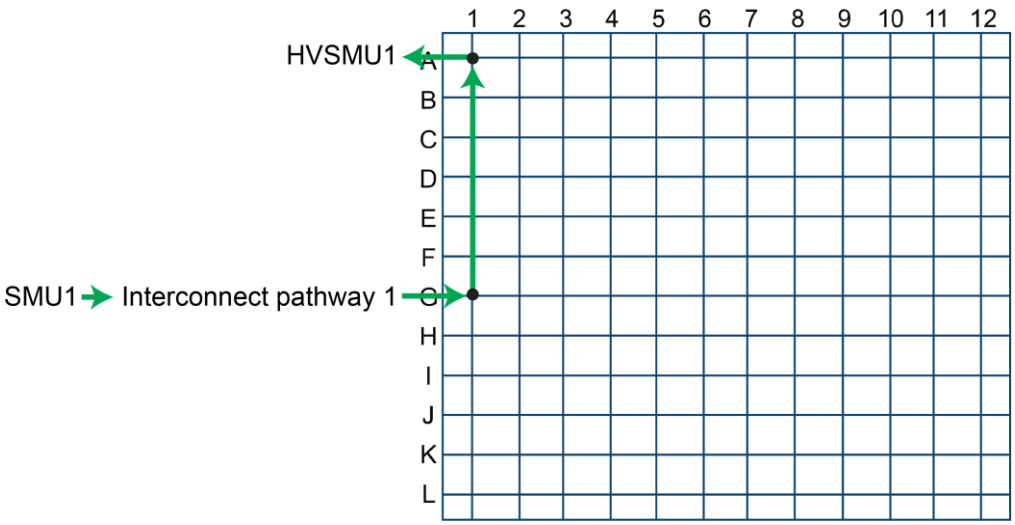
This tests verifies that the connections are good between the HVM1212A matrix and the 7531 switch cards in the 707B matrix mainframe.

Interconnect pathway test sequence:

1. For interconnect pathway 1, SMU1 connects to pin 1 and HVSMU1 connects to pin 1.
2. SMU1 forces 10 mA and HVSMU1 forces 0 V.
3. HVSMU1 measures current and compares it to the value forced by SMU1.
4. HVSMU1 forces 10 mA and SMU1 forces 0 V.
5. SMU 1 measures current and compares it to the value forced by HVSMU1.
6. Steps 1 through 5 are repeated on each additional interconnect pathway.

Result: If the measured value is within 10 percent of the forced value, the interconnect pathway passes. If the measured value is not within 10 percent of the forced value, the interconnect pathway fails.

Figure 258: HVM1212A matrix interconnect pathway test



HVM1212A matrix coil relay test

The HVM1212A matrix coil relay test is an internal self-test of the HVM1212A. The test verifies that current is actually flowing through the coil of each relay in the system. The test runs on each crosspoint in the matrix, and if any crosspoint fails, the diagnostics software returns a failure.

HVM1212A matrix discharge relay test

The HVM1212A matrix discharge relay test is an internal self-test of the HVM1212A. On each column of the HVM1212A matrix, there is a discharge relay. When a column is not used, the discharge relay is connected to ground through a 10 MΩ resistor. This test verifies that the discharge resistor is working properly.

SMU range tests

The SMU range tests are used to verify that each source-measure unit (SMU) can force and measure properly on each of the available voltage and current ranges. The following topics describe these tests.

Voltage and current range tests

The range test uses two SMUs: The forcing SMU (SMU under test) and the measuring SMU.

The `$KIDAT/3kV_spec_limits.ini` file contains a list of source-measure unit (SMU) models supported and their voltage and current ranges. Each current and voltage range is tested at ±90% and ±10% of range.

High-voltage SMU range test

NOTE

The diagnostics software overrides any `MAX_VOLTAGE` settings you may have specified in the `icconfig_<QMO>.ini` file.

High-voltage SMU range test sequence:

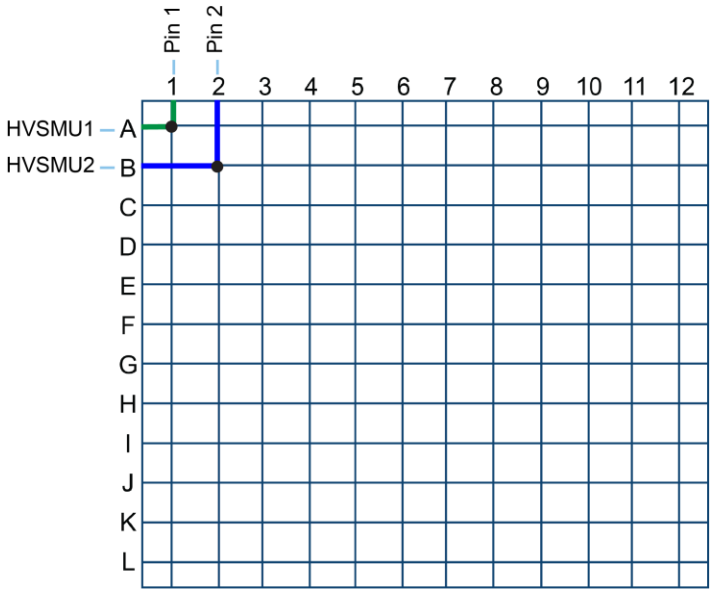
1. HVSMU1 and HVSMU2 are set to a supported range (3000 V maximum for the 2657A measuring HVSMU).
2. HVSMU1 forces the appropriate value and then measures itself.
3. HVSMU2 also measures the value.
4. The two measurements are compared against the limits specified in the `$KIDAT/3kv_spec_limits.ini` file for the range.
5. This sequence is repeated for each additional supported range on this pair of SMUs.
6. Steps 1 through 5 are repeated with HVSMU2 as the forcing SMU and HVSMU1 as the supporting SMU.

Result: If the readings are within the limits, the test passes. If the readings are outside the limits, the test fails.

The test order for a range is as follows: +10%, +90%, –10% –90%.

The following figure shows the matrix connection used to test the ranges for HVSMU1. HVSMU2 is used as the measuring SMU for this test.

Figure 259: High-voltage SMU range test



Low-voltage SMU range test

Source-measure unit (SMU) range test sequence:

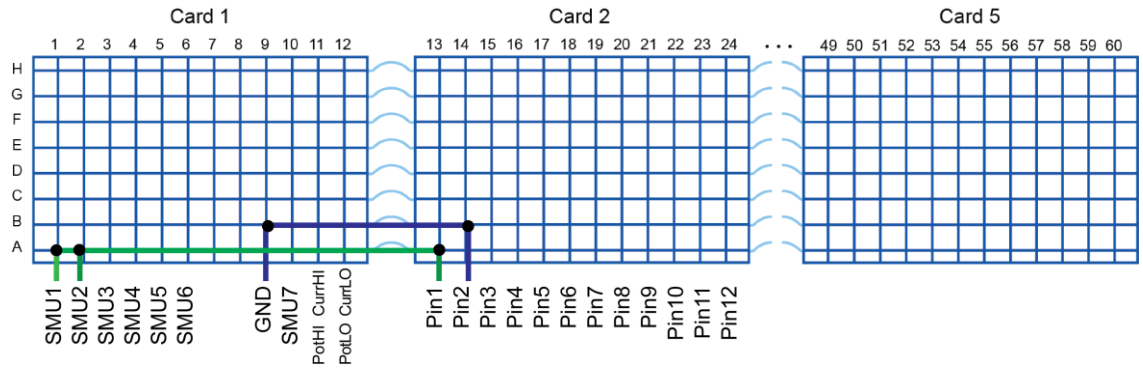
1. Each SMU is set to the appropriate range (200 V maximum for the 2636B measuring SMU).
2. The forcing SMU forces the appropriate value and then measures itself.
3. The supporting SMU also measures the value.
4. The two measurements are compared against the limits specified in the `$KIDAT/diag_limits.ini` file for the range.
5. This sequence is repeated for each additional supported range on this pair of SMUs.
6. Steps 1 through 5 are repeated using a different pair of SMUs (for example, SMU3 and SMU4).
7. The test continues until all ranges have been tested on all SMU combinations.

Result: If the readings are within the limits, the test passes. If the readings are outside the limits, the test fails.

The test order for a range is as follows: +10%, +90%, –10% –90%.

The following figure shows the matrix connection used to test the ranges for SMU1. SMU2 is used as the measuring SMU for this test.

Figure 260: SMU range test connection



Tests for other instruments

The diagnostics software can perform tests for other instruments: Capacitance meters, oscilloscopes, pulse generators, digital multimeters, and prober chucks. The following topics describe these tests.

Capacitance meter test

The diagnostics software does different procedures for CMTR1 and CMTR2.

CMTR1 test sequence:

1. The diagnostics software makes an impedance measurement across an open and a short. If the ratio of these measurements is less than 1000, the test fails.
2. Tests the capacitance-voltage unit (CVU) biasing capability by forcing 3 V and measuring with a functional SMU. If the measured value is within 2 percent of the forced voltage, the test passes; if it is not, the test fails.
3. Forces –0.5 V and repeats the test.

Oscilloscope test

The oscilloscope (SCP) test uses the built-in self-test function of the 4200-SCP2HR card.

Pulse generator test

The diagnostics software performs the following test on the pulse generator (PGU).

Pulse generator test sequence:

1. Creates a 4 V pulse and measures with a source-measure unit (SMU).
2. Creates a –4 V pulse and measures with a SMU.
3. If the measured value is within ± 1% of the pulse height, the test passes; if not, the test fails.

Digital multimeter test

Digital multimeter (VMTR) test sequence:

1. The diagnostics software forces voltage on a known functional source-measure unit (SMU).
2. Measures the voltage with the VMTR. If the measured voltage is within ± 2% of the forced value, the test passes; if not, the test fails.
3. The test is repeated with the SMU forcing negative voltage.

Prober chuck connection test

The prober chuck connection test verifies that the connection to the chuck is a valid Kelvin connection. The test sequence is similar to the one described in Kelvin pin tests (on page 12-18), except only the chuck pin is tested.

NOTE

This test is valid only if the chuck has been defined in the `icconfig_<QMO>.ini` file.

This test runs automatically at the end of the diagnostics test sequence, but can also be run separately if needed.

System verification

The system verification routines test and measure the instruments in the test system to ensure that they are operating within the system specifications.

The diagnostics routines described in the previous section must be executed before the system verification tests can run. You must verify that the system is operational before you try to validate the specifications.

The result of the system verification test is pass or fail. If the test fails, you can use the diagnostics test results to troubleshoot the system.

NOTE

It is possible that the diagnostics test passes, but the system verification fails. This is because the system verification routine uses tighter acceptance limits for the tests. For the verification process to be valid, all instruments in the system must be within their calibration cycle, and not overdue for calibration.

The S540 system is tested during manufacturing to make certain it conforms to its warranted specifications (as stated on its data specification sheet). Each system is calibrated using measurement standards traceable to the International System of Units (SI) through the National Institute of Standards and Technology (NIST), or other National Metrology Institutes such as National Physical Laboratory (NPL), Physikalisch-Technische Bundesanstalt (PTB), and others. The S540 meets the requirements of standard ANSI/NCSL Z540-1-1994: Calibration Laboratories and Measuring Test Equipment – General Requirements.

The system verification procedure compares each source-measure unit (SMU) in the system to every other SMU using appropriate measurement practices. Assessment of the accuracy procedure is done through measurement uncertainty evaluations, accounting for known major contributors, and ensuring failing decision risks are minimized.

An S540 that passes the system verification procedure conforms to its warranted specifications as stated on its data specification sheet if the individual instruments in the system are calibrated within the recommended annual calibration interval.

Keithley recommends annual calibration of the individual instruments and offers on-site calibration services. For more information, contact your local Keithley representative.

Verification probe card

The system verification routines require the use of a specific probe card that is included when ordering the 9140A-PCA probe card adapter with the S540 system. This probe card has pins shorted together to create pathways from the system to the probe card and back to the system. The pathways are used to verify system performance against the system specifications.

WARNING

Hazardous voltages may be present on the probe card adapter, even after you disengage the interlock. Cables can retain charges after the interlock is disengaged, exposing you to live voltages that, if contacted, may cause personal injury or death.

Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

The verification probe card shown in the following figure shorts pin 1 to pin 2, pin 3 to pin 4, pin 5 to pin 6, and so on, up to the number of pins available in the system. This allows the system verification routines to connect various instruments to the appropriate pins for the test sequence. The system verification software verifies that this card is installed before starting the test sequence.

NOTE

The number and configuration of jumper wires on the verification probe card are determined by the system configuration.

The following images show examples of a shorted verification probe card and a blank probe card.

Figure 261: Shorted verification probe card for S540 systems

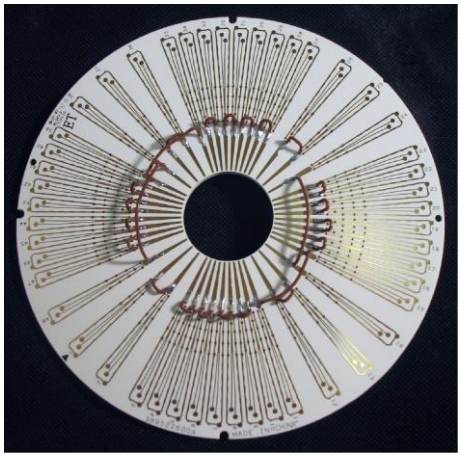
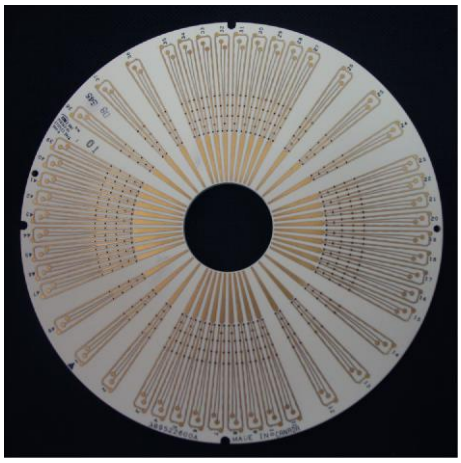


Figure 262: Blank probe card for S540 systems



Verification tests to execute

The tests used to verify the system specifications are similar to those used for diagnostics testing. The main difference is that the pathways include a round-trip to and from the probe card through the shorted pins, and the verification measurement data is compared against a different set of limits than the diagnostics tests. See Diagnostic tests (on page 12-21) for test details. See the product specifications for your system for the limit values.

Matrix leakage tests

System verification runs matrix leakage tests that are similar to the diagnostics tests, except that no quasistatic capacitance measurements are made. See the Matrix tests (on page 12-22) topic for more information.

The diagnostic tests described in Diagnostic tests (on page 12-21) are done against looser limits than the verification tests. Verification tests use the standards stated in the system performance specifications.

SMU verification

This test connects the force source-measure unit (SMU) to pin A and measures SMU to pin B (where pin A and pin B are shorted together). It then executes the diagnostics SMU range tests against tighter limits than the limits used for diagnostic testing.

One SMU pair tests all of the pin-pair combinations. Once this test is completed, a known functional pin pair is used to test all of the other SMU pairs.

This test executes in the following phases:

- 3 kV verification phase
- Phase 1 tests one SMU pair (usually SMU1 and SMU4) through all matrix pathways
- Phase 2 tests all SMU pairs through one pathway
- Phase 3 tests one SMU pair through each shorted pin pair

NOTE

To save time, the phase 3 test does not test each SMU range; it only tests the highest and lowest SMU voltage and current ranges.

Troubleshooting

WARNING

Hazardous voltages may be present within the S540 cabinet that can cause personal injury or death due to electric shock. Only personnel who are qualified to perform troubleshooting on electrical systems and instrumentation should use this information and perform troubleshooting on the S540 system.

This section provides important information about how to troubleshoot the hardware under failure conditions. The content is provided to help you resolve any issues you may encounter during prerequisite and diagnostics testing. If you do not see the specific failure you are having under the test categories, please contact your local Keithley Instruments field service engineer (FSE).

The troubleshooting information is based on the tests completed during prerequisite and diagnostic testing. Test messages and errors or failures are displayed in the Status and Result tabs of the diagnostic software user interface (see the User interface (on page 12-3) topic for more information).

NOTE

You may want to save the results of prerequisite and diagnostic tests in a log text file for future reference when troubleshooting. To save the Status and Result logs to a `.txt` file, right-click each window and select **Save**. In the Status and Result tabs, failures display in red text; the text color does not transfer to the saved text file.

During execution of a test, the Status and Result tabs provide information pertaining to executed tests. Any time you want to interrupt the automatic scrolling during an active diagnostics session, click the window and scroll. If you want the automatic scrolling to resume, press **Ctrl** and **End** on your keyboard.

Troubleshooting overview

Diagnostics testing gives you information to determine if the system is working properly. When there is a problem, the Result and Status tabs usually tag failed tests or show a summary of a failed operation with the word "FAILURE."

Because the S540 consists of many instruments connected in a system, a failure in a test cannot always be correlated to a problem with a specific instrument. Single test failures are not returned as simple error numbers or codes; but as the result of a measurement or condition that does not meet the criteria to pass. To compensate for this, many of the tests are repeated with the same conditions, but on different combinations of instruments or matrix pathways. This provides a variety of results to identify an overall behavior or pattern.

For example, testing a source-measure unit (SMU) range is done with two SMUs: One that is sourcing (the SMU being tested, SMUA) and another that is measuring (SMUB). If the test result is outside of the expected limits, the test fails. In this case, a single test result is returned and it is not clear which SMU caused the problem. The test continues using other SMUs in the system to identify the problematic SMU.

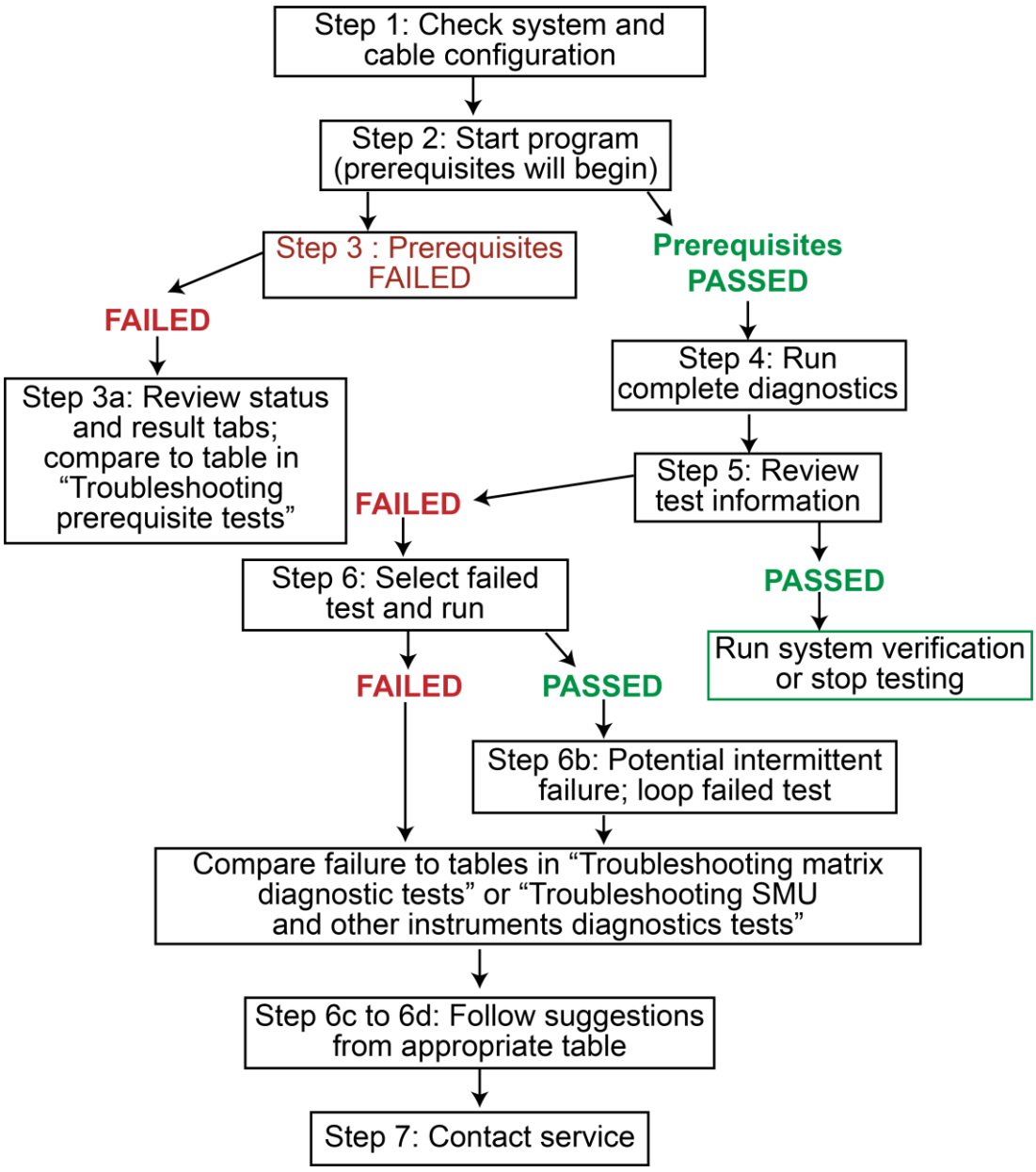
To continue this example, assume SMUA has a problem. During the test, SMUA is used both as a source SMU and as a measure SMU. But other SMUs are also used to measure SMUA. If two or more SMUs only have failures when sourcing or measuring with SMUA, then SMUA is confirmed the problem SMU. By reviewing the results involving SMUA interaction with other SMUs in the system, it is possible to determine if SMUA is the common instrument in the various failures. If multiple SMUs show failures only when interacting with SMUA, then SMUA is confirmed as not working correctly.

This means that the troubleshooting situation requires the system user to review the Status and Result information and match the pattern of test results to the problem situations provided in the tables below. This type of pattern matching, or deduction, is key to effectively troubleshooting the S540 system.

Troubleshooting flow chart

The troubleshooting flow chart represents the procedures that you should use. Refer to the Troubleshooting procedure steps (on page 12-49) after this flow chart for detailed, step-by-step information.

Figure 263: S540 troubleshooting flow chart



Troubleshooting procedure steps

To use the diagnostics software to troubleshoot problems, follow the steps described below.

CAUTION

To prevent damage to the pins on your shorted probe card, use the blank probe card (supplied with your system) when troubleshooting.

Step 1: Check system connections and start the diagnostics program

1. Ensure that the system and cabling are properly configured.

   Separate the probe card from the wafer. For a Kelvin (4-wire) system, each force-measure pair should be connected. No force pins should be connected to other force pins.

2. Start the diagnostics program (see The diagnostics process (on page 12-1) for more information).

   The diagnostics program automatically runs a set of prerequisite tests.

Step 2: Review prerequisite test results

1. If the prerequisite tests return `PASS`, run the complete diagnostics routine (see The diagnostics process (on page 12-1) for detailed information). If the prerequisite tests return `FAIL` (do not pass):
   - Review the Status and Result tab information and compare it to the information in Troubleshooting prerequisite tests (on page 12-50) to determine possible causes.
   - Right-click the **Status** tab and **Result** tab and save the log information into separate files.
   - If specific causes cannot be determined, review cabling and connection information in the *S540 Power Semiconductor Test System Administrative Guide* to ensure proper configuration.
2. If the prerequisite tests continue to fail, contact your local Keithley field service engineer (FSE) for additional guidance. Provide the failure information, failure frequency, and any saved test logs.
   - Contact your local Keithley Instruments field service engineer (FSE) to explain the prerequisite failure details.
   - Provide the Status and Result log files.
   - If directed by a Keithley FSE, review Shadow log file (on page 12-60) for more information.

Step 3: Run the complete diagnostics routine

Once the prerequisite tests have passed, you can run the complete diagnostics routine.

NOTE

Generally, failures are listed in the summary section at the end of each test. Scrolling through the Status or Result tabs shows individual test failures, usually in red text. Keithley recommends that you run the complete diagnostics routine when troubleshooting an issue for the first time.

1. Review the Status and Result tabs for test result information.
2. Find the specific test that results in the failure. If there is more than one failure, address the first failure that appears in the Status or Result log.
3. Right-click the Status and Result tab and save the log information to separate files for each test. Include the test date and time in the filename so that you can compare any future files.
4. After saving the contents of the Status and Result tabs, clear the information in the Status and Result tabs. This makes it easier to troubleshoot the next test results, if necessary.

Step 4: Rerun only the failing tests

1. Find and enable the test that had the error. Execute only the selected test.
   - Review the Status and Result tabs for test result information. Determine if the same error occurred by comparing this most recent test with the earlier complete diagnostics test.
   - If the error does not recur, the failure condition may be intermittent.
2. Use the loop feature to repeat one or more tests to obtain additional results about the failure.
   - If the error is the same, review the appropriate tables for additional information.
   - For matrix-related failures, see Troubleshooting matrix diagnostic tests (on page 12-53) for specific conditions.
   - For SMU, CMTR, VMTR, scope, or PGU failures, see Troubleshooting SMU and other instruments diagnostic tests (on page 12-57) for specific conditions.
   - After following the suggestions in the appropriate table, rerun the appropriate diagnostics test.
3. If the failure persists, contact your local Keithley field service engineer (FSE) for additional guidance. Provide the failure information, failure frequency, and any saved test logs.
   - Contact your local Keithley Instruments FSE to explain the prerequisite failure details.
   - Provide the Status and Result log files.
   - If directed by a Keithley FSE, review the Shadow log file (on page 12-60) for more information.

Troubleshooting prerequisite tests

The following tables are presented in chronological test order and represent the tests that are done during prerequisite diagnostic testing. They contain guidance on the failures that may occur during the prerequisite diagnostic tests. If you cannot find a comparable solution to your issue, contact your local Keithley Instruments field service engineer (FSE).

2-wire force side prerequisite test troubleshooting

Test failures or symptoms	Possible solutions

A failing SMU.
 	
LPT error with code E0020 or E0029 on the Result tab.
 	
Example errors:
 	
`>>>> Tester LPT Error: 2017/08/14 12:24:01 – E0020 (unknown) Command not executed because a previous error was encountered.`
 	
`>>>> Tester LPT Error: 2017/08/14 12:24:01 - E0029 (unknown) An error generated by instrument = 4102, code = 1211, message = Node 2 is inaccessible.`
	1. Check power to instruments:
	■ Open the front door of the system.
	■ Check the front panel of the SMU.
	■ If the front panel is dark:
	- Check the power button
	- Check the power plug cable connection to the SMU rear panel (see the power distribution unit (PDU) information in the "Equipment startup" section of the *S540 Power Semiconductor Test System Administrative Guide*)
	2. Check the TSP connection on the master 2636B SMU:
	■ Ping the master SMU.
	Use the system IP address and adding `.2` to the end of IP#. This IP address is hardcoded (not the customer IP address).
	PC: 192.186.1.1
	Master SMU: 192.168.1.2
	707B: 192.186.1.5
	4200A-SCS: 192.168.1.12
	
	Example: At a Microsoft® Windows® command prompt or Linux® terminal, ping `192.168.1.2`.
	■ If no connection is found, visually check the TSP connection on the rear panel of the 2636B SMU.
	■ If the cable is connected, do the following:
	a. Exit diagnostics user interface.
	b. Reset the router (located in rear of system).
	c. Reset all SMUs.
	d. For KTE (Linux® systems only):
	- If the terminal shows a prompt, send `stop_ic.pl`, then send `run_ic.pl`.
	- Type `Ctrl` and run above `.pl` calls.
	e. Run the diagnostics user interface.

2-wire force side prerequisite test troubleshooting (continued)

Test failures or symptoms	Possible solutions

A failing SMU.
 	
Failure on SMU for all pathways while operating as a forcing SMU and a measuring SMU.
 	
Example: SMU2 failed
 	
`FAILURE (2-Wire) Forcing with SMU1 and measuring with SMU2 using pathway A`
…
`FAILURE (2-Wire) Forcing with SMU2 and measuring with SMUs using pathway B`
	1. Check the Result tab for LPT error code E0020 or E0029. If this error occurred, see the troubleshooting information in the above section.
	2. Check the HI Sense and HI Force cable connections on the rear panel of the failing SMU; verify cables are correctly connected to the appropriate terminals.
	3. Check the LO Sense connection to LO Patch Panel for the failing SMU.
	To check these items, open the rear door of the system. The cables are labeled with the SMU number and function.

A failing matrix card pathway.
 	
Failure seen for same pathway on most SMU combinations.
 	
Example: Pathway E
 	
`FAILURE (2-Wire) Forcing SMU1 and measuring with SMU4 using pathway E`
…
`FAILURE (2-Wire) Forcing SMU3 and measuring with SMU7 using pathway E`
	A closed relay in the pathway. If intermittent, you may pass this test and then fail the next time.
	To check for this, close the diagnostics user interface and restart it. Repeat the test a few times to determine if the relay closure is intermittent or if the relay is stuck.

Kelvin prerequisite test troubleshooting

Test failures or symptoms	Possible solutions

Failure with all SMUs on the same pin and matrix path for Kelvin detection.
 	
Result is not enough Kelvin connections found.
 	
Example: Relay 3C02 stuck open
 	
`Using SMU1 for Kelvin detection.`
`FAILURE!! SMU1: Pin 02 (ForceCol=26 [Slot=3 Col=2] SenseCol=26 [Slot=3 Col=2]) Matrix Path: C Hi,Lo=9.91000e+37`
…
`Using SMU8 for kelvin detection.`
`FAILURE!! SMU8: Pin 02 (ForceCol=26 [Slot=3 Col=2] SenseCol=26 [Slot=3 Col=2]) Matrix Path: C Hi,Lo=9.91000e+37`
`ERROR! We need at least two pins to be non-adjacent.`
	Issue with relay on matrix card.
	
	Contact your local Keithley Instruments field service engineer (FSE) for guidance with this issue.
	
	■ If directed by the FSE, remove power from the system using the "Remove system power" procedure in the *S540 Administrative Guide*.
	■ If directed by the FSE, remove the suspect card from the matrix using the "General replacement procedure" information in the *S540 Administrative Guide* and any additional information provided by the FSE.
	
	NOTE: All pins in the system must have good Kelvin connections. The card type is returned in the Result tab as part of the prerequisite testing information.

Not enough Kelvin connections found.
 	
Example:
 	
`ERROR! We did not find two valid pins...`
…
`ERROR! We need at least two pins to be non-adjacent.`
	1. Verify that all Kelvin pin connections on the probe card are good.
	2. Check ground cable connections (GND Sense and GND Force) from the LO-patch panel to the matrix card.
	
	NOTE: All pins in the system must have good Kelvin connections. The card type is returned in the Result tab as part of the prerequisite testing information.

Troubleshooting matrix diagnostic tests

The following tables provide information about failures that may occur during matrix diagnostic testing. If you cannot find a comparable solution to your issue, contact your local Keithley Instruments field service engineer (FSE).

Force-side relay test troubleshooting

Test failures or symptoms	Possible solutions

Force-side failure on a second card.
 	
Example:
 	
`FAILURE SMU1 1A01,1B09, 3A02,3B02 Relay 3A02 and/or 3B02 failed to close`
`RETEST: Relay 3A02 and/or 3B02 failed to close Relay 3A02 failed to close`
`RETEST: Relay 3A02 failed to close Relay 3B02 failed to close`
`RETEST: Relay 3B02 failed to close`
	A faulty relay:
	
	1. If intermittent, this test may pass, then fail the next time. To check for this, update the loop count field, select **Only this test** from the Matrix tab and run the selected test.
	2. Relay could be stuck open or closed, as demonstrated by the failures. The card must be replaced.
	
	NOTE: An inspection of the card should be done by a qualified service representative.

Matrix leakage test troubleshooting

Test failures or symptoms	Possible solutions

Failing SMU1 when measuring capacitance. It is on the edge of failing.
 	
Example: Showing failure when Guard is lost on SMU1 Force Hi and Sense Hi
 	
`FAILURE Using SMU1 and path B: Measured capacitance of 5.0331e-11 Limit=5e-11 %Limit=100.663 Card 1 only`
…
`SUCCESS Using SMU1 and path E: Measured capacitance of 4.9881e-11 Limit=5e-11 %Limit=99.7623 Card 1 only`
…
`FAILURE Using SMU1 and path F: Measured capacitance of 5.0346e-11 Limit=5e-11 %Limit=100.693 Card 1 only`
	1. Verify SMU1 cable connections are valid from the rear panel of the SMU to the matrix.
	2. If the connections are okay, there may be a bad cable, adapter, or connector.
	a. Replace SMU1 cables (Sense HI, Force HI, and Sense LO).
	b. Move the SMU1 cable.

Failing SMU1 measured capacitance for all pathways.
 	
Example: Showing failure when SMU1 Sense Hi is disconnected
 	
`FAILURE Using SMU1 and path A: Measured capacitance of 1.8512e-09 Limit=5e-11 %Limit=3702.44 Card 1 only`
`FAILURE Using SMU1 and path A: Measured capacitance of 1.8844e-09 Limit=8e-11 %Limit=2355.5 Cards 1 and 2`
`FAILURE Using SMU1 and path A: Measured capacitance of 1.7612e-09 Limit=8e-11 %Limit=2201.5 Cards 1 and 3`
	1. Verify SMU1 cable connections are valid from the rear panel of the SMU to the matrix.
	2. If the connections are okay, there may be a bad cable, adapter, or connector.
	a. Replace SMU1 cables (Sense HI, Force HI, and Sense LO).
	b. Move the SMU1 cable.

Matrix continuity test troubleshooting

Test failures or symptoms	Possible solutions

Failing on all pathways.
 	
Example: Showing failure when SMU1 Sense Hi is disconnected
 	
`FAILURE Forcing with SMU1: SMU2 measured 10.8346 ohms on path A. Limit=2 %Limit=541.73 Card 1 only`
`FAILURE Forcing with SMU1: SMU2 measured 10.7649 ohms on path A. Limit=2 %Limit=538.246 Cards 1 and 2`
`FAILURE Forcing with SMU1: SMU2 measured 10.779 ohms on path A. Limit=2 %Limit=538.948 Cards 1 and 3`
	1. Verify SMU1 cable connections are valid from the rear panel of the SMU to the matrix.
	2. If the connections are okay, there may be a bad cable, adapter, or connector.
	a. Replace SMU1 cables (Sense HI, Force HI, and Sense LO).
	b. Move the SMU1 cable.

Kelvin connection pin leakage test troubleshooting

Test failures or symptoms	Possible solutions

One or more pins fail the Kelvin check.
 	
Example:
 	
`FAILURE Testing Pins: 1, 2. Forcing with SMU1 on path A: SMU2 measured -7.73668e-14 on path B. Limit=1e-08 %Limit=0.000773668`
`SMU1 using path A is not forcing voltage of 50!(Possible short to GND?) Measured=0.0967503 Error limit=0.05`
 	
`SUCCESS Testing Pins: 1, 2. Forcing with SMU1 on path B: SMU2 measured 9.99991e-10 on path A. Limit=1e-08 %Limit=9.99991`
 	
`FAILURE Testing Pins: 2, 3. Forcing with SMU1 on path A: SMU2 measured -2.0597e-12 on path B. Limit=1e-08 %Limit=0.020597`
`SMU1 using path A is not forcing voltage of 50!(Possible short to GND?) Measured=0.0967741 Error limit=0.05`
 	
`SUCCESS Testing Pins: 2, 3. Forcing with SMU1 on path B: SMU2 measured 9.99997e-10 on path A. Limit=1e-08 %Limit=9.99997`
 	
`Running Pin Guard Tests`
`FAILURE Using SMU1 to test pin 1. Measured capacitance of 2.46724e-06. Limit=1.5e-10 %limit=1.64483e+06`
`FAILURE Using SMU1 to test pin 2. Measured capacitance of 9.02001e-08. Limit=1.5e-10`
	A faulty relay:
	
	1. If intermittent, this test may pass, then fail the next time. To check for this, update the loop count field, select **Only this test** from the Matrix tab and run the selected test.
	2. Relay could be stuck open or closed, as demonstrated by the failures. The card must be replaced.
	
	NOTE: An inspection of the card should be done by a qualified service representative.

Troubleshooting SMU and other instrument diagnostic tests

The following tables contain guidance on the failures that may occur during the diagnostic tests on source-measure units (SMUs), capacitance meters (CMTRs), and pulse cards (PGUs). If you cannot find a comparable solution to your issue, contact your local Keithley Instruments field service engineer (FSE).

SMU voltage range test troubleshooting

Test failures or symptoms	Possible solutions
A SMU or SMUs fail on a specific range or value.	A faulty relay: 1. If intermittent, this test may pass, then fail the next time. To check for this, update the loop count field, select **Only this test** from the Matrix tab and run the selected test. 2. Relay could be stuck open or closed, as demonstrated by the failures. The card must be replaced. **NOTE:** An inspection of the card should be done by a qualified service representative.

SMU current range test troubleshooting

Test failures or symptoms	Possible solutions

SMU measurement out of range due to additional low resistance in series.
 	
Example: Additional resistance on SMU1 force
 	
`ERROR: SMU2 measurement out of range: low=0.3819 high=0.4181`
`ERROR: SMU1 measurement out of range: low=0.3819 high=0.4181`
`FAILURE Range 1A:`
`SMU2 forced 0.4 measured 0.166343`
`SMU1 measured 0.166614`
`Error=-0.2334`
`Limit=0.0181`
`%Limit=-645`
	1. Verify SMU1 cable connections are valid from the rear panel of the SMU to the matrix.
	2. If the connections are okay, there may be a bad cable, adapter, or connector.
	a. Replace SMU1 cables (Sense HI, Force HI, and Sense LO).
	b. Move the SMU1 cable.

CMTR test troubleshooting

Test failures or symptoms	Possible solutions
Bias test failure on all path pairs (A & B, C & D, and so on) due to swapped HI and LO signal connections.	Check the capacitance meter (CMTR) HI Pot and HI Cur connections to LO Pot and LO Cur cables. Verify that the connections to CMTR and CMTR protection module are correct.

Pulse card test troubleshooting

Test failures or symptoms	Possible solutions

SMU measurement error.
 	
Example:
 	
`FAILURE PGU1B Pulsed 4 and SMU1 measured -4.21634`
`FAILURE PGU1B Pulsed -4 and SMU1 measured -4.27092`
`OR`
`FAILURE PGU1B Pulsed 4 and SMU1 measured -0.3397`
`FAILURE PGU1B Pulsed -4 and SMU1 measured -0.335039`
`OR`
`FAILURE PGU1A Pulsed 4 and SMU1 measured 1.98491`
`FAILURE PGU1A Pulsed -4 and SMU1 measured -1.98648`
	1. Check failed pulse generator unit (PGU) channel output connection from rear of the S4200 PGU to matrix card connection.
	For example, `PGU1B` indicates PGU pulse card 1, channel 2
	2. There may be some resistive load on the failed PGU channel output.

Advanced troubleshooting information

The following topics contain advanced troubleshooting information.

Intermittent failures

Some failures may be intermittent. The looping feature (see the Control area (on page 12-5) topic for more information) of the diagnostics software is useful to repeat all or some diagnostic tests to look for occasional failures.

Shadow log file

The shadow log file is created by the diagnostics software. This file contains more information than is available in the Status or Result tab areas of the user interface. If the Status or Report tab content does not provide enough information for your needs, you can refer to the shadow log file.

The file name is `shadow.log`. This file always contains the latest information. When this file size reaches 1.5 MB, the content is saved into a backup shadow log file (for example, `shadow1.log`) and a clean `shadow.log` file is used. There can be up to five backup log files before the information is removed.

The location of this file:

- **On a Windows® system using ACS version 5.4** is `C:\ACS\Diagnostics\Report\`
- **On a Linux® system using KTE version 5.8** is `$KILOG` directory (default = `/opt/kiS540/log`)

IC log file

For Linux® KTE systems, there is a log file for the instrument control (IC) server process, which may be useful for certain troubleshooting situations. The IC process log file is `$KILOG/ic_<QMO>_YYYYMMDD_HHMM.log` file, where:

`<QMO>` is the QMO number of the tester.

`YYYYMMDD_HHMM` is the date and time when the IC process was started.

Section 13

Maintenance

In this section:

Introduction

This section describes routine maintenance that should be done on the S540 System.

NOTE

Replacement of system hardware should be done by qualified service personnel only.

Repair and replacement

Keithley Instruments offers a fee-based service agreement with all S540 systems. Under this agreement, a field service engineer will either repair or replace equipment. For more information about this service agreement, contact Keithley Instruments at 1-800-935-5595.

For additional information about specific parts, operations, and maintenance of Keithley instruments, refer to the documentation for the instrument for details before attempting to replace or repair any equipment. You can download manuals for related instruments at tek.com/keithley.

Adjustment

Keithley Instruments recommends annual adjustment of the individual instruments in your system and offers this as an on-site service. A field service engineer (FSE) will adjust instrumentation and perform system verification according to the warranted system specifications. For more information about adjustment or other S540 services, contact your sales representative.

You can also do system verification as described in the *S540 Reference Manual* (part number S540-901-01).

WARNING

Hazardous voltages may be present on the probe card adapter, even after you disengage the interlock. Cables can retain charges after the interlock is disengaged, exposing you to live voltages that, if contacted, may cause personal injury or death.

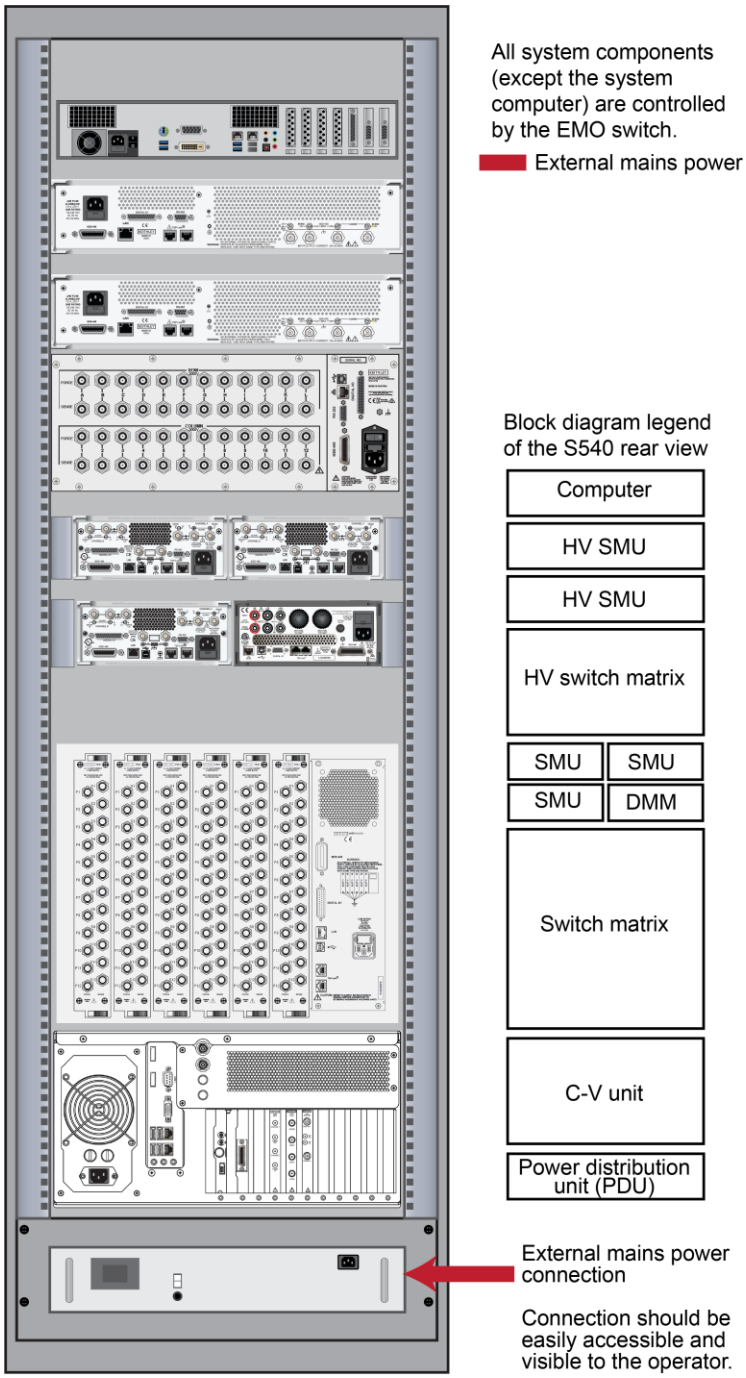
Never attempt to touch or change a probe card when tests are running. You must be absolutely certain that all tests have stopped before making contact with anything in the vicinity of the probe card adapter. Also, never run tests without a probe card installed.

For information about instrument-level adjustment, refer to documentation for each of the instruments in the system (available on the Product Support web page (tek.com/product-support)).

Power distribution and emergency off

The following figure contains a simplified example layout of various components in the S540 system.

Figure 264: S540 system cabinet rear view



Data hub license

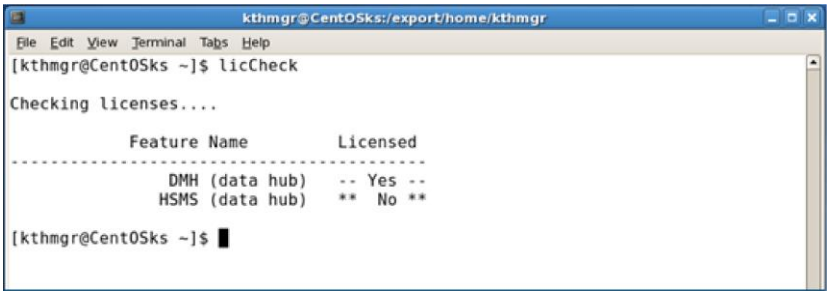
The data hub license allows the Keithley Test Environment (KTE) software to communicate with a prober. This license is installed on all systems when they are shipped.

If you upgrade or reinstall the KTE software, you may need to reinstall the data hub license.

To upgrade systems with KTE version 5.8.0 and later:

1. Open a new Linux® terminal session.
2. Type `licCheck` and press **Enter** to check if the datahub license is installed correctly.

Figure 265: Check datahub license installation



3. If it is not installed correctly, you will need to edit the file at `$KIHOME/.ki_setup`, where `$KIHOME/.ki_setup` is usually `opt/kiS530/.ki_setup`.
4. Go to line 33 in the file and change it to `setenv COMPUTERNAME q1234` (where `1234` is your QMO number).
5. Reboot the computer to reinitialize the datahub.

Decommissioning an S540 test system

The S540 Power Semiconductor Test System does not contain any intentionally released substances, but may contain substances that are potentially hazardous to the environment if not properly recycled.

For example, systems produced before July 22, 2017 and shipped into the European Union may contain lead (Pb) as a part of the solder to connect electronic components and system interconnects. A list of possible hazardous substances is in European Union Directive “Restricting the use of hazardous substances in electrical and electronic equipment” (RoHS) Directive 2011/65/EU or later. This list includes:

- Lead (Pb)
- Mercury (Hg)
- Cadmium (Cd)
- Hexavalent chromium (Cr6+)
- Polybrominated biphenyls (PBB)
- Polybrominated diphenyl ether (PBDE)
- Bis (2-ethylhexyl) phthalate (DEHP)
- Butyl benzyl phthalate (BBP)
- Dibutyl phthalate (DBP)
- Diisobutyl phthalate (DIBP)

For more detailed information, see the European Union Directive.

To minimize environmental impact at system end of life, treat the system, cables and connections, and all subassemblies as waste electrical and electronic equipment (WEEE) category 9. Reference the European Union Directive on waste electrical and electronic equipment (WEEE Directive 2012/19/EU or later).

Follow these directives to minimize environmental impact at any location in the world. Always follow all local, state, and country environmental laws; these take priority over other directives.

Take all product and subassemblies to a reputable electronics recycle company for proper recycling. Several approved recyclers are identified on the Tektronix website at tek.com.

Any cleaning solutions used during the life of the system (such as isopropyl alcohol (IPA)) should be disposed of separately and properly.

The S540 system may be used with other equipment such as automatic probers and third-party probe card adapters. Tektronix is not in control of these products and the original equipment manufacturer should always be contacted for proper recycling procedures.

Index

C

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W

X

Z

Keithley Instruments
Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-800-935-5595 • www.tek.com/keithley

KEITHLEY
A Tektronix Company