

# S530 Parametric Test System

## KTE Software Manual

S530-921-01 Rev. B / January 2014



S530-921-01

A Greater Measure of Confidence



# S530 Parametric Test System

## KTE Software

### Manual

©2011-2014, Keithley Instruments, Inc.

All rights reserved.

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

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

Document Number: S530-921-01 Rev. B / January 2014

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

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

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

The types of product users are:

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

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

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

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

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

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

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

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

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

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

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

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

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


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


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

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

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


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


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

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

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

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

The  symbol indicates a connection terminal to the equipment frame.

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

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

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

Instrumentation and accessories shall not be connected to humans.

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

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

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

Safety precaution revision as of January 2013.

# Table of Contents

Section	Topic	Page
1	<b>Introduction</b> .....	1-1
	Overview of the manual .....	1-2
	Manual contents .....	1-2
	Other documentation .....	1-3
	Manual addenda .....	1-3
	System software .....	1-3
2	<b>Creating Test Plans</b> .....	2-1
	Introduction .....	2-2
	Wafer Description Utility (WDU) .....	2-3
	WDU main window .....	2-3
	Wafer Setup window .....	2-6
	Target Setup window .....	2-8
	Probe Pattern Editor window .....	2-9
	Site Editor window .....	2-12
	Site Optimization window .....	2-14
	Wafer Graph Editor window .....	2-14
	User Defined Values window .....	2-21
	Test Structure File Editor (TSE) .....	2-23
	TSE main window .....	2-23
	Creating a test structure file .....	2-25
	Keithley Interactive Test Tool (KITT) .....	2-27
	KITT main window .....	2-27
	KITT results window .....	2-30
	KITT Math, Array and Logic Expression Support .....	2-31
	Math .....	2-31
	Array Support .....	2-32
	Using Generated Identifiers in a Test Macro .....	2-35
	KTE data usage .....	2-36
	Use of ibupu in KITT .....	2-38
	Parameter Set Editor (PSE) .....	2-38
	PSE main window .....	2-38
	Keithley User Library Tool (KULT) .....	2-41
	KULT main window .....	2-41
	Parameter window .....	2-45
	Description window .....	2-47
	Command line interface .....	2-49
	Copying user libraries with kult_copy_lib .....	2-50
	Migrating user libraries with migrate_usrlib .....	2-51
	Locking a module .....	2-51
	Changing library attributes .....	2-52
	KTE library locking .....	2-54
	Run-time library locking .....	2-54
	Edit-time library locking .....	2-54
	Troubleshooting .....	2-55
	Limits File Editor (LFE) .....	2-55
	LFE main window .....	2-56
	Limits data entry area .....	2-57
Limits editor dialog window .....	2-59	
Keithley Test Plan Manager (KTPM) .....	2-60	

Wafer plan builder .....	2-60
Wafer plan builder main window .....	2-60
Title bar .....	2-61
Wafer plan builder file menu .....	2-61
Wafer plan builder options menu .....	2-62
Wafer plan builder help menu .....	2-62
Wafer description file selection field .....	2-62
Limits file selection field .....	2-62
Probe card file selection field .....	2-63
Plan type selection button .....	2-63
Sort subsites button .....	2-63
Probe patterns/site plans list .....	2-63
Test macros/site plans list .....	2-63
Site plan/wafer plan builder field .....	2-63
Insert buttons .....	2-63
New site plan button .....	2-64
Wafer plan file description field .....	2-64
Wafer description file description field .....	2-64
Cassette plan builder .....	2-64
Title bar .....	2-65
Cassette plan builder file menu .....	2-65
Cassette plan builder options menu .....	2-66
Cassette plan builder help menu .....	2-66
File description field .....	2-66
Execution engine selection field .....	2-66
Probe card file selection field .....	2-66
Wafer description file selection field .....	2-66
Lot ID data field .....	2-66
Global data file selection field .....	2-66
Slot ID type selection buttons .....	2-67
Wafer test plan field .....	2-67
Standard UAP file field .....	2-67
UAP module/KTM area .....	2-67
Test documentation tool .....	2-67
Main window .....	2-67
Using the test documentation tool .....	2-69
Generating documentation during testing .....	2-71
Adaptive testing .....	2-71
Overview .....	2-71
Zone based testing .....	2-71
Result based testing .....	2-73
KTE Support Utilities .....	2-81
Random Pattern Generation (rand_pat) .....	2-81
File filtering .....	2-83
<b>3 Test Execution .....</b>	<b>3-1</b>
Introduction .....	3-2
Keithley Operator Interface Editor (KOPED) .....	3-2
KOPED main window .....	3-3
KOPED file menu .....	3-3
KOPED edit menu .....	3-4
KOPED options menu .....	3-4
KOPED help menu .....	3-4
Pop-up menus .....	3-4
Keithley Test Execution Engine (KTXE) .....	3-4
KTXE main window .....	3-5
The execution process .....	3-8
User Library files required for KTXE execution .....	3-15
The KTXEErrorHandler function .....	3-15
Error and event logging .....	3-16
Lot suspend/resume .....	3-17
Multi-cassette Multi-lot Testing Utility (multi_cassette) .....	3-17
Wafer id usage in KTXE .....	3-17
Abort flags .....	3-18
KTXE results and their structures .....	3-21

	.....	3-22
<b>4</b>	<b>Data Analysis</b> .....	4-1
	Introduction .....	4-2
	Data output formats .....	4-2
	Preview .....	4-2
	Keithley Summary Utility (KSU) .....	4-2
	Summary reports .....	4-3
	Standard report .....	4-4
	Raw report .....	4-6
	KSU description .....	4-7
	Starting KSU .....	4-7
	KSU main window .....	4-7
	Other KSU main window controls .....	4-14
	Command-line options for KSU .....	4-15
	Keithley Curve Analysis Tool (KCAT) .....	4-16
	KCAT main window .....	4-16
	Scaling window .....	4-18
	Data window .....	4-19
	Pop-up tools menu .....	4-19
	KDFtoKCS File Conversion Utility .....	4-20
<b>5</b>	<b>System Administration</b> .....	5-1
	Introduction .....	5-2
	Workstation applications .....	5-2
	Keithley tool palette .....	5-2
	System configuration .....	5-4
	ktpm.ini file .....	5-6
	Environment variables .....	5-7
	Log file environment variables .....	5-10
	File management .....	5-11
	KCM utility description .....	5-12
	Project environments .....	5-12
	Select tester .....	5-14
	Addtester .....	5-16
	System customization .....	5-17
	System integration .....	5-19
	Linear Parametric Test Library (LPTLIB) .....	5-19
	Keithley Data Files (KDF) library .....	5-19
	Keithley User Interface (KUI) library .....	5-20
<b>A</b>	<b>KDF</b> .....	A-1
	Test data logging .....	A-2
	Keithley Data Files (KDF) library .....	A-2
	Using limits files .....	A-3
	Programming examples .....	A-6
	Data logging routines .....	A-11
	PutLot .....	A-11
	PutWafer .....	A-12
	PutSite .....	A-12
	PutParam .....	A-13
	PutParamList .....	A-13
	EndLot .....	A-14
	EndWafer .....	A-14
	EndSite .....	A-15
	GetLot .....	A-15
	GetWafer .....	A-15
	GetSite .....	A-16
	GetParam .....	A-17
	GetParamList .....	A-17
	GetLotData .....	A-18
	MatchParam2Limit .....	A-18
	FileExist .....	A-18
	LotExist .....	A-19
	GetStartTime .....	A-19

DeleteLot .....	A-19
DeleteWafer .....	A-19
DeleteSite .....	A-20
DeleteParam .....	A-20
DeleteLimitCode .....	A-20
DeleteLimit .....	A-21
Update comment routines .....	A-21
GetComment .....	A-21
PutComment .....	A-21
Update limits routines .....	A-21
GetLimitCode .....	A-21
GetLimit .....	A-22
PutLimit .....	A-22
Structure handling routines .....	A-23
AddNewLimitCode	
AddNewLot	
AddNewWafer	
AddNewSite	
AddNewParam .....	A-23
CreateNewLmtCode	
CreateNewLot	
CreateNewWafer	
CreateNewSite	
CreateNewParam .....	A-23
FindFirstLmtCode	
FindFirstLot	
FindFirstWafer	
FindFirstSite	
FindFirstParam .....	A-24
FindLastLimitCode	
FindLastLot	
FindLastWafer	
FindLastSite	
FindLastParam .....	A-25
FindNextLimitCode	
FindNextLot	
FindNextWafer	
FindNextSite	
FindNextParam .....	A-25
FindPrevLimitCode	
FindPrevLot	
FindPrevWafer	
FindPrevSite	
FindPrevParam .....	A-26
InsertNewLmtCode	
InsertNewLot	
InsertNewWafer	
InsertNewSite	
InsertNewParam .....	A-26
RemoveLimitCode	
RemoveLot	
RemoveWafer	
RemoveSite	
RemoveParam .....	A-27
LimitExist .....	A-28
KDF user tag data .....	A-28
Example of use within a Cassette Plan .....	A-30
UAP points .....	A-30
Structure definitions .....	A-31
Sample programs .....	A-31
Logging one PARAM at a time, Data Retrieval through Get routines	A-31
Logging a Linked List of PARAMs, Data Retrieval using GetLotData	A-34
<b>B Keithley User Interface .....</b>	<b>B-1</b>
Introduction .....	B-2



	User interface constants.....	B-2
	User interface library variables.....	B-3
	User interface library functions.....	B-4
	GetProgramArgs — Get Program Command Line Arguments.....	B-4
	InitUINew — Initialize User Interface Library.....	B-5
	InputMsgDlg — Input Message Dialog.....	B-6
	LotDlg — Lot Information Dialog.....	B-6
	OkCancelAbortMsgDlg — Ok Cancel Abort Message Dialog.....	B-7
	OkCancelMsgDlg — Ok Cancel Message Dialog.....	B-7
	OkMsgDlg — Ok Message Dialog.....	B-7
	QuitUI — Quit User Interface.....	B-7
	ScrollMsgDlg — Scrollable Message Dialog.....	B-7
	ScrollMsgDlgClr — Scrollable Message Dialog Clear.....	B-8
	ScrollMsgDlgMsg — Scrollable Message Dialog Message.....	B-8
	StatusDlg — Status Dialog.....	B-8
	UpdateModelessDlgs — Update Modeless Dialogs.....	B-9
	UpdateStatusDlg — Update Status Dialog.....	B-10
	VarMsgDlg — Variable Message Dialog.....	B-10
	WfrldsDlg — Multiple Wafer Information Dialog.....	B-11
	WfrldDlg — Single Wafer Information Dialog.....	B-12
	YesNoAbortMsgDlg — Yes No Abort Message Dialog.....	B-13
	YesNoCancelMsgDlg — Yes No Cancel Message Dialog.....	B-14
	ContSkipAbortDlg — Continue Skip Abort Message Dialog.....	B-14
	LBoxDlg — List Box Message Dialog.....	B-14
	KTE KUI localization.....	B-15
<b>C</b>	<b>KTE File Formats.....</b>	<b>C-1</b>
	Wafer description file format.....	C-3
	Test structure file format.....	C-6
	Keithley test macro (.ktm).....	C-7
	Probe card file format.....	C-10
	Global data file format.....	C-11
	Parameter set file format.....	C-12
	Parameter limits file format.....	C-14
	Wafer test plan file format.....	C-16
	Cassette test plan file format.....	C-18
	Keithley data file format.....	C-21
	User access point file format.....	C-23
	Keithley plot file format.....	C-24
	KULT module file format.....	C-25
<b>D</b>	<b>Data Pool.....</b>	<b>D-1</b>
	Data pool.....	D-2
	Advanced data pool use.....	D-11
	Description of the data pool functions.....	D-12
	Recommendations, hints, and examples.....	D-15
<b>E</b>	<b>User Access Points (UAPs).....</b>	<b>E-1</b>
	User Access Points (UAPs).....	E-2
	User access point usage.....	E-3
	Types of UAPs.....	E-4
	UAP locations.....	E-5
	Usage of LPTLIB functions at UAPs.....	E-5
	Distributed user libraries.....	E-5
	Test macro debugging.....	E-6

# 1

## Introduction

---

This section gives an introduction to the KTE software manual. The information covered in this section is as follows:

- **Overview of the manual** — Describes the contents of this manual and provides references for other sources of information.
- **Safety** — Details safety concerns that should be observed during system operations.

## Overview of the manual

This manual details the operation and use of the KTE software for the S530 Parametric Test System. It explains the different software tools used in creating and executing test programs and how system files are controlled and handled.

## Manual contents

- **1 - Introduction** — Gives an overview of the manual and an overview of the entire KTE software system.
- **2 - Creating Test Plans** — Discusses the tools used to create test plan files.
- **3 - Test Execution** — Discusses the tools that control the selection and execution of cassette plans.
- **4 - Data Analysis** — Describes the tools used to gather and analyze results data provided by an executed test program.
- **5 - System Administration** — Details the setup of the KTE software, file management, system configuration, and Ethernet specifications.
- **Appendix A - Keithley Data Files (KDF) Library** — Describes the set of routines used to organize and save parametric test data into ASCII data files.
- **Appendix B - Keithley User Interface (KUI) Library** — Describes the different subroutines that support the creation of sophisticated operator interface dialogs for test programs.
- **Appendix C - KTE Tool File Structure Examples** — Provides file structure examples for all of the files created using the KTE software.
- **Appendix D - Data Pool** — Describes the data pool which is used to hold global data while KTXE is running.
- **Appendix E - User Access Points (UAPs)** — Provides descriptions and examples on how to use User Access Points.
- **Appendix F - KTE Localization** — Describes localization which allows the KTXE interface to be displayed in any language.

## Other documentation

Although this manual provides some detailed information about the KTE software system, it does not repeat information contained in other manuals. For information pertaining to the KTE software, that is not contained within this manual, refer to the following documentation:

- **Linear Parametric Test Library (LPTLIB) Manual** — A library of subroutines that provide direct program control of the instrumentation and matrix.
- **Prober Library Manual** — A library of subroutines used to control a variety of manual and automatic wafer probers.

## Manual addenda

Any improvements or changes concerning the product or manual will be explained in an addendum. Addenda are provided in a page replacement format. Simply replace the obsolete pages with the new pages where indicated.

## System software

This version of the software differs from earlier versions of the KTE software tools. Earlier versions (V2.X) of the KTE software created compiled, uneditable, executable files. This version of the software creates a much more flexible, editable test plan. The benefits derived from this version include:

- Test plans that can be edited to suit the needs of the user.
- User Access Points (UAPs) that allow the user to extend the functionality of the test execution engine.
- Test execution is data driven. This means the test plan execution is derived from data files read by the test execution engine.

[Figure 1-1](#) shows a diagram of how KTE is used to create and execute a cassette plan. [Figure 1-1](#) step numbers (from 1 through 8b) are keyed to [Figure 1-3](#).

The data elements for a test plan are created in much the same way as they were for the executable files in earlier versions of KTE:

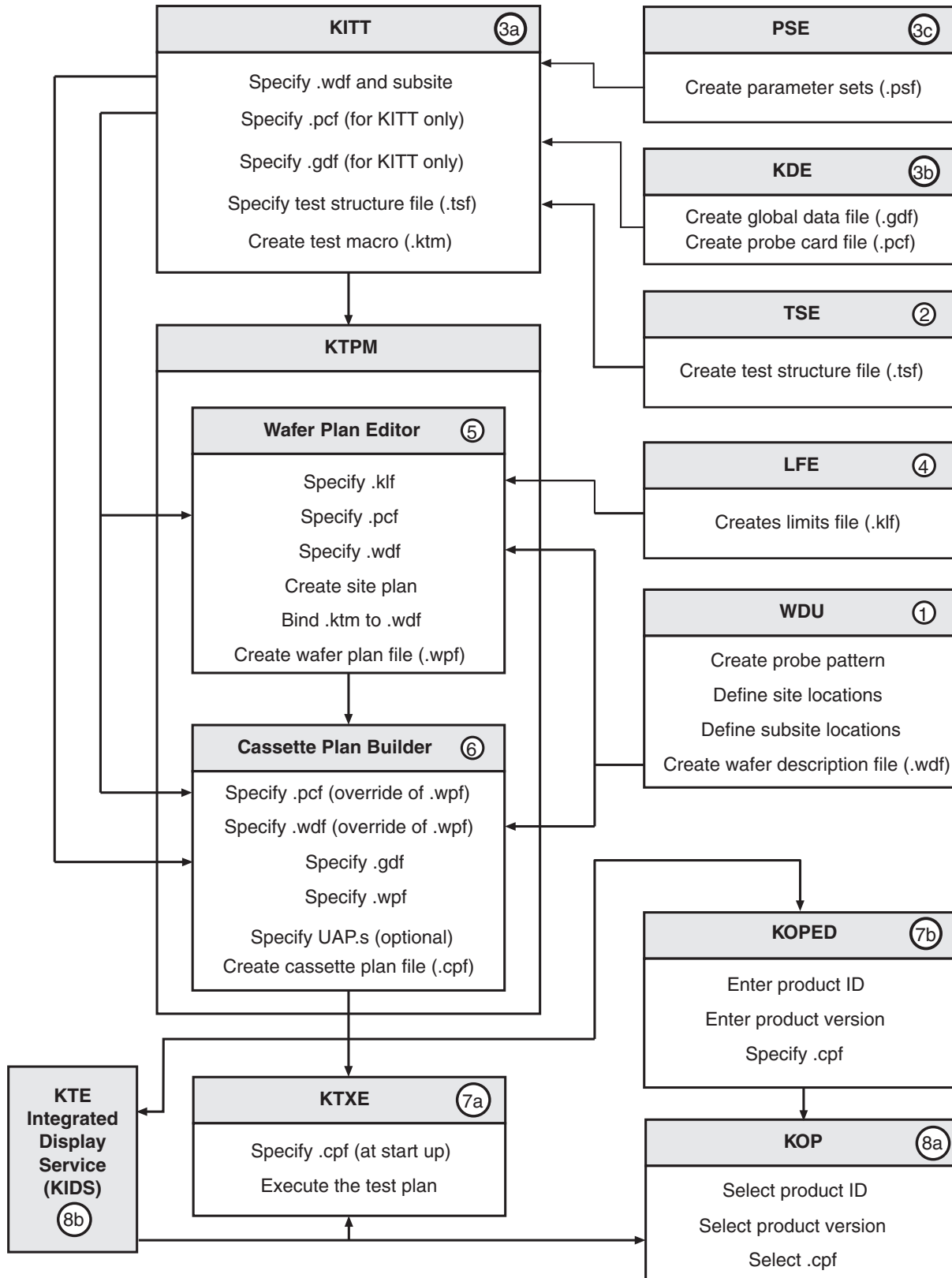
- The Wafer Description Utility (WDU) is used to describe the wafer and specify the test sites and subsites.
- The Limits File Editor (LFE) is used to specify the test result limits.
- The Keithley User Library Tool (KULT) is used to create user libraries that are accessible in KITT.
- The Keithley Interactive Test Tool (KITT) is used to create the test macro that will be run on the selected wafer.

The improvements that have come as a result of this version include:

- Adding the Keithley Data Editor (KDE), which gives the user access to all probe card files (.pcf), global data files (.gdf) and pre-defined identifiers (PDI) from one centralized location.
- Adding the Test Structure File Editor (TSE), which allows the user to create a set of test parameters for each specific device that is to be tested.

- Adding the Parameter Set Editor (PSE), which allows the user to edit/add parameter set data for a specific routine.
- Changes to the KITT Parameter Entry Window that allow the user to provide more detailed parameter data.
- 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 (.wfp) that specify the wafer description, site and subsite plans, the test macros that will be applied to the wafer, and the limits file that will 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.
- Adding the ability to use Math expressions in test macros and conditional execution of test modules.

Figure 1-1  
Cassette plan creation flow diagram



The data files created using KTE all play important roles in creating a coherent test plan. [Figure 1-2](#) shows a diagram describing the data contained within each file and the placement of each file when a test plan is created.

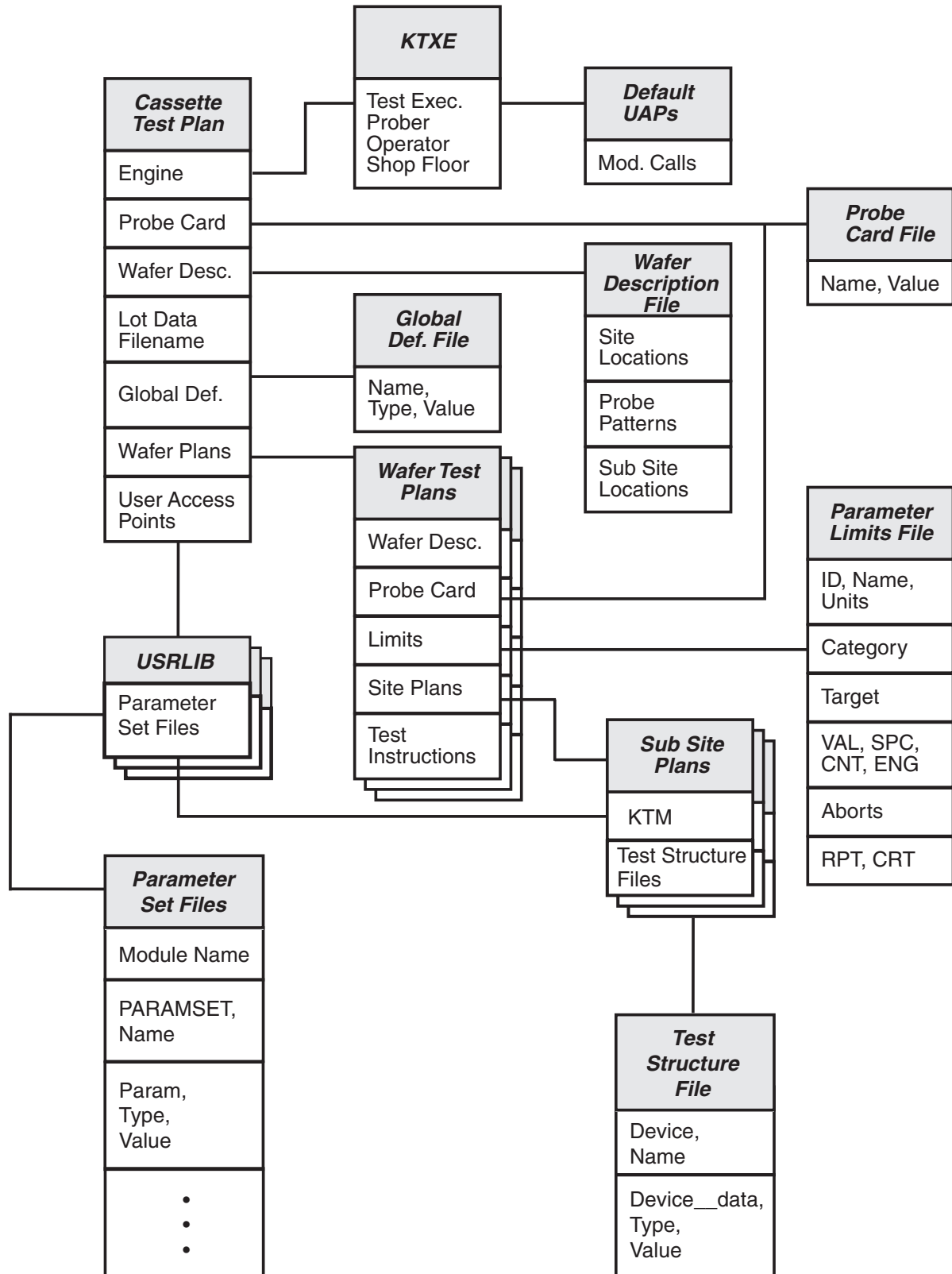
During test plan creation, it is possible to specify the probe card file and the wafer description file in more than one place. If this occurs, the file specified in the Cassette Test Plan takes precedence over the file specified in the Wafer Test Plan.

The 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 will be ignored if specified in the wafer plan file.

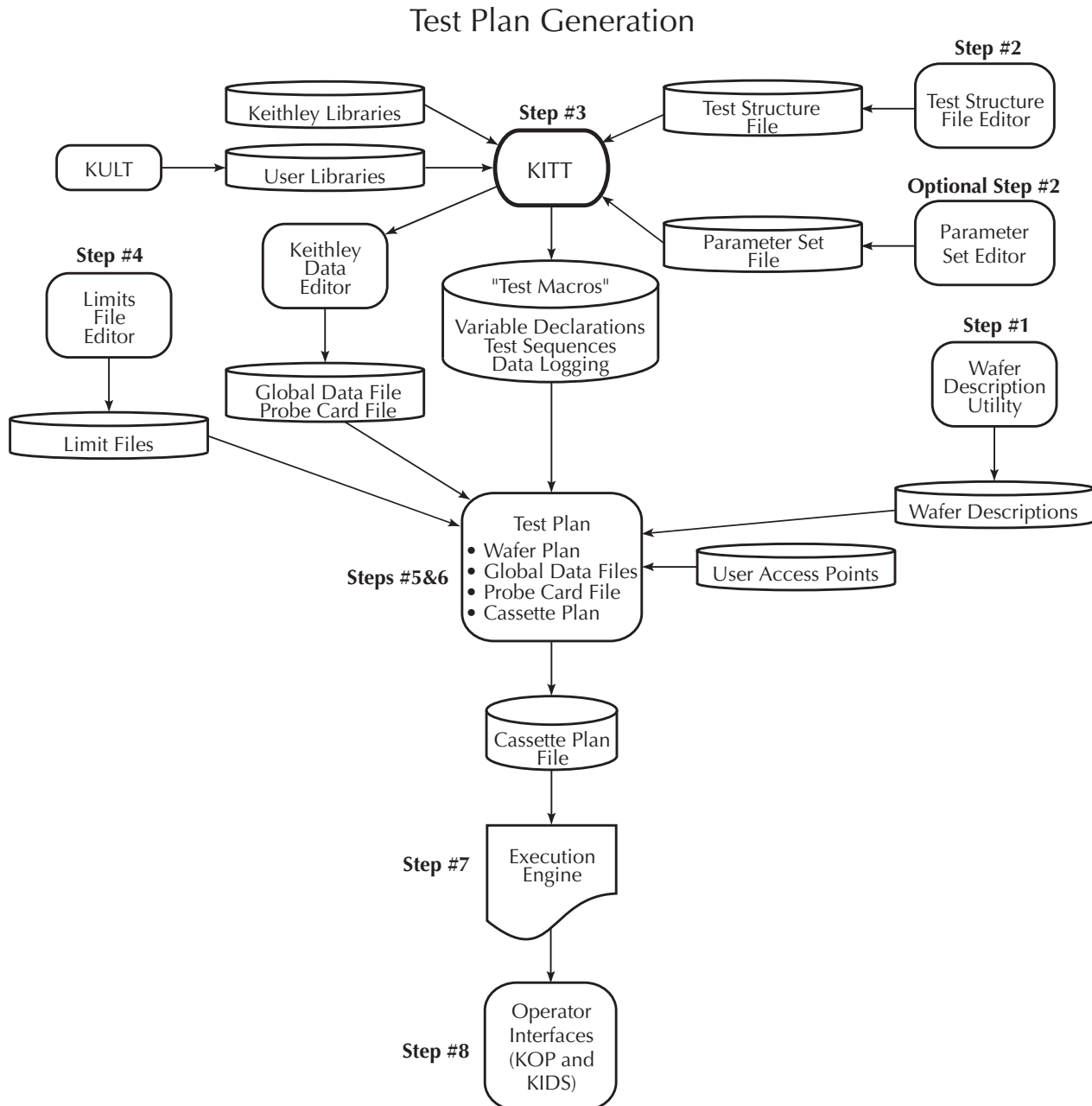
Figure 1-2  
Data file hierarchy





Production-quality test plans are created using KTPM, the Keithley Test Plan Manager. 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 various pre-built program elements and allow KTPM to “bind” them together to produce the final cassette plan. [Figure 1-3](#) shows the test plan creation process.

Figure 1-3  
**Test plan generation block diagram**



The software that is used by the system is the Keithley Test Equipment (KTE) Software platform. With the KTE software tools you will be able to create and execute the test programs that will be performed by the parametric test system. The KTE software tools contain:

**Wafer Description Utility (WDU)** — Captures information to describe a wafer's size, orientation, sites, and subsites.

**Test Structure File Editor (TSE)** — Used to create test structure files that contain device specific parameters.

**Keithley Interactive Test Tool (KITT)** — Used to create test macros (including the data used), and interactively debug the macros on the tester.

**Keithley Data Editor (KDE)** — Used to create/modify global data files and probe card files.

**Parameter Set Editor (PSE)** — Used to edit/add parameter for a library module routine.

**Keithley User Library Tool (KULT)** — Used to create test modules and user libraries that can be used in KITT and/or as UAPs to control the testing process.

**Limits File Editor (LFE)** — Used to capture information describing the criteria for judging the measured parameters identified in KITT.

**Keithley Test Plan Manager (KTPM)** — Used to create wafer test plans, cassette plans, and cassette plan documentation.

**Keithley Operator Interface Editor (KOPED)** — Used to create the operator's cassette plan selection tree that will be presented to the operator in KOP.

**Keithley Operator Interface (KOP)** — Used by the operator to select and execute a cassette plan.

**Keithley Test Execution Engine (KTXE)** — Executes the cassette plans created using KTPM. The KTXE used is specified in the cassette plan.

**KTE Integrated Display Service (KIDS)** — Used for recipe selection, execution, display status, and 300mm Automation option (if installed).

**Keithley Summary Utility (KSU)** — Used to view the results, comparing them against the parameter limits entered in LFE.

**Keithley Curve Analysis Tool (KCAT)** — Used to create a graphical representation of array results data generated from a test run using KITT.

For more information regarding the KTE software tools, refer to Sections 2, 3, 4, and 5 of this manual.

# 2

## Creating Test Plans

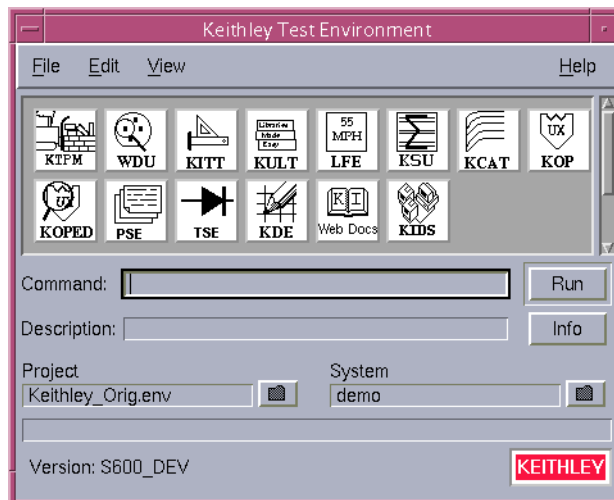
---

## Introduction

This section covers the tools required to create the files used when building a test plan. In this section the following will be discussed:

- **Wafer Description Utility (WDU)** — This section will describe how to use WDU to create a wafer description file that describes the wafer used during the testing process.
- **Test Structure Editor (TSE)** — This section will describe how to use TSE to create parameters for a specified device located at a specific wafer subsite.
- **Keithley Interactive Test Tool (KITT)** — This section will describe how to use KITT to create a test macro containing the tester commands.
- **Parameter Set Editor (PSE)** — This section will describe how to use PSE to modify parameter set data assigned to a library module.
- **Keithley User Library Tool (KULT)** — This section will describe how to use KULT to create a user library that contains test modules which can be used when creating a test macro.
- **Limits File Editor (LFE)** — This section will describe how to use LFE to create a limits file containing parameter limits that are compared to the results acquired during testing.
- **Keithley Test Plan Manager (KTPM)** — This section will describe how to use KTPM to create a cassette plan file (.cpf) that can be executed by KTXE.
- **Keithley Tool Pallet (KTP)** — The tool palette provides a point and click interface into the KTE development environment. The tool palettes for the S530 system, shown in [Figure 2-1](#), contain an icon for each of the test plan development and utility functions. Refer to the *System Administration* section for further details about KTP.

Figure 2-1  
Keithley tool palette



When you have completed this section, you will be able to utilize WDU, KITT, KULT, and LFE to create files necessary to the test plan building process.

## Wafer Description Utility (WDU)

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 through seven different windows within WDU:

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

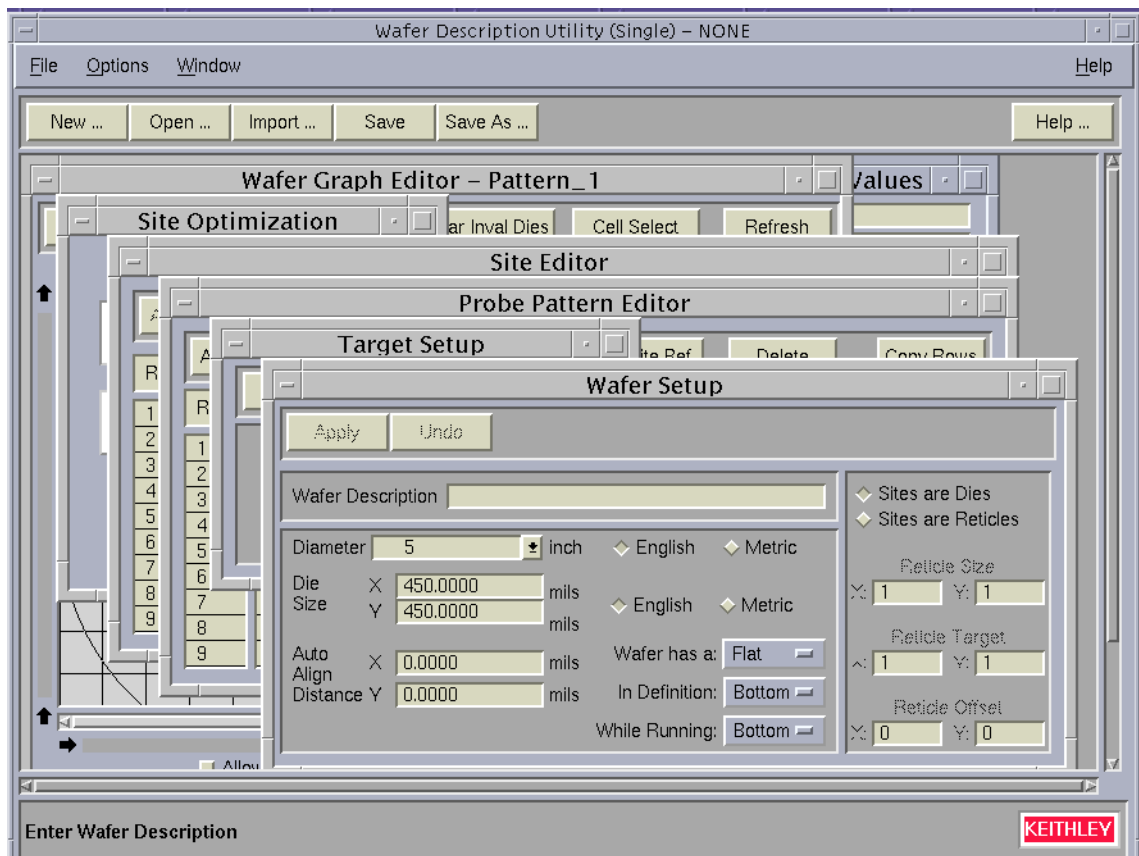
The resulting file is saved in an easy-to-read ASCII format.

### WDU main window

The WDU main window appears when WDU is initiated.

A description of the main window components follows.

Figure 2-2  
WDU main window



A description of the main window components follows:

**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.

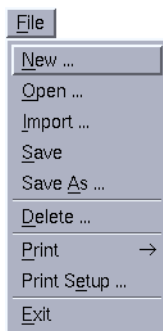
**Control buttons** — The control buttons (New, Open, Import, Save, Save As, and Help) give you control over the information in the main window work area.

**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 [Figure 2-3](#).

*Figure 2-3*  
**WDU File menu**



The menu items include the following:

**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. Import script must be specified in the wdu.ini file. See WDU on-line 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.

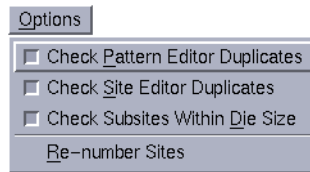
**Print Setup** — Lets you set up the printed page parameters.

**Exit** — Exits WDU.

## WDU Options menu

Selecting Options from the menu bar produces the menu in [Figure 2-4](#).

*Figure 2-4*  
**WDU Options menu**



The menu items are as follows:

**Check Pattern Editor Duplicates** — If selected, 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** — If selected, checks for duplicate subsite IDs and for duplicate site IDs when saving the file.

**Check Subsites Within Die Size** — If selected, verifies that all subsite coordinates are within the die size when saving the file. WDU uses the absolute values of subsite coordinates in its comparison against the die size.

**Re-number Sites** — If selected, re-numbers site IDs. Any existing site IDs will be replaced with the default names, Site\_N, where N is an increasing number. This will allow the user to re-order site names/numbers. The user is prompted before any action is taken.

## WDU Window menu

Selecting Window from the menu bar produces the menu in [Figure 2-5](#).

*Figure 2-5*  
**WDU Window menu**



The menu items are as follows:

**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 WDU.

**About** — Displays the installed WDU version number.

## Control buttons

Clicking on a control button performs one of the following procedures:

**New...** — Allows the selection of Single or Multi-project mode, and initializes 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 WDU on-line 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 on-line help information about WDU.

## Main window work area

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

**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 site(s).

**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.*

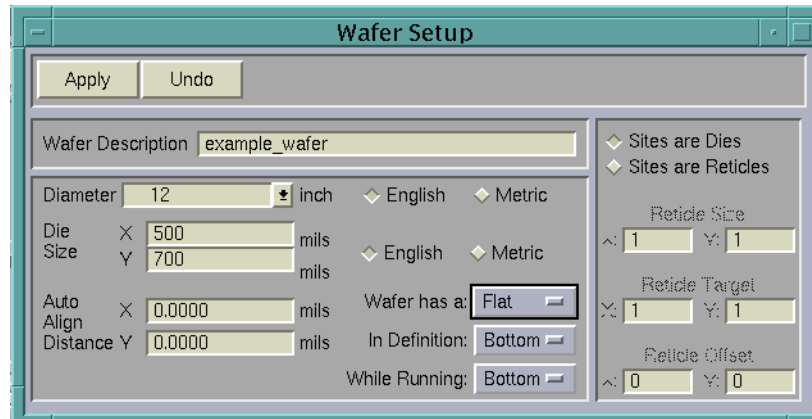
The functions of each window are discussed in detail in the subsections that follow.

## Wafer Setup window

The Wafer Setup window, [Figure 2-6](#), is used to enter all the wafer size, die size, site definition, and auto align distance data.



Figure 2-6  
**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 on the down arrow and making the desired selection.

**Die Size** — Lets you enter the X and Y die size (mils or millimeters). These fields are not used for multi-project 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/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/etc. button)** — Specifies the normal flat or notch orientation.
- **“While Running” (Bottom/Top/etc. 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 three sets of fields described below specify the size and coordinates of the reticle.

**Reticle Size field** — When Sites are Reticles is selected (see above), 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 (see above), 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 (see below), 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 (see above), 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, [Figure 2-7](#), sets the X,Y coordinates of the target site (die or reticle) and the direction for increases in the X,Y coordinate values.

Figure 2-7  
**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 multi-project wafers (multi-project wafers have more than one repeating reticle pattern).

### X dir/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, [Figure 2-8](#), is used to specify site probe patterns during testing. These site probe patterns will be paired with tests in the 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 will be 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. The user can change this default name by clicking on the site name and entering the desired name for the site.

Figure 2-8  
Probe Pattern Editor window

Row	Pattern Name/Site Name	X Offset	Y Offset
1	Pattern_1		
2	Site_4	0	0
3	Site_13	-4	0
4	Site_15	-8	0
5	Site_16	4	0
6	Site_18	8	0
7	Site_23	0	3
8	Site_24	0	6
9	Site_25	0	-3
10	Site_26	0	-6
11	Pattern_2		

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 Previous 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

There are six control buttons along the top of the Probe Pattern Editor window. These buttons function for both single and multi-project files. The function of each button is as follows.

**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.

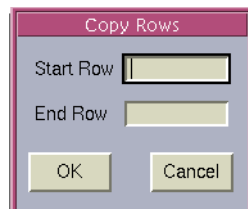
**Add Site Bef.** — Adds a site name field to the site listing before the currently selected field. Site names are not necessary for single project wafers because all sites are identical.

**Delete** — Deletes the currently selected field.

**Copy Rows** — Will copy the rows indicated in the Copy Rows window, [Figure 2-9](#), after the row where the cursor is currently positioned.

Figure 2-9

### 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 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.

**Multi-project** — Each site within a pattern must be given a name because a multi-project pattern is not restricted to a single, repeating reticle frame. Many probe patterns are also supported in a multi-project wafer file.

## X and Y offset fields

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

**Single project** — The X,Y offset values are based on site (die or reticle) coordinates. The coordinate values are set by the user through the Target Setup window.

**Multi-project** — The X,Y values provided are position information in millimeters or mils. These values are absolute position information. The wafer alignment could be used to put the wafer in the correct initial position.

## Site Editor window

The Site Editor window, [Figure 2-10](#), is used to specify the different sites and their subsites. The information required depends on the project type selected.

*Figure 2-10*  
**Site Editor window**

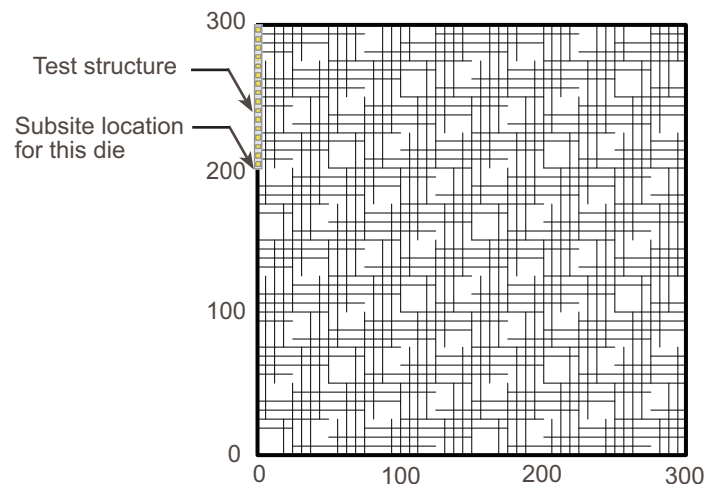


*Single-project wafers* have only one site type. Therefore, no site information is necessary, only subsite information. *Multi-project wafers* can have more than one site type. Therefore, both the site's description and subsite locations must be provided.

Specify subsite X,Y coordinates as follows:

- 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 [Figure 2-11](#).

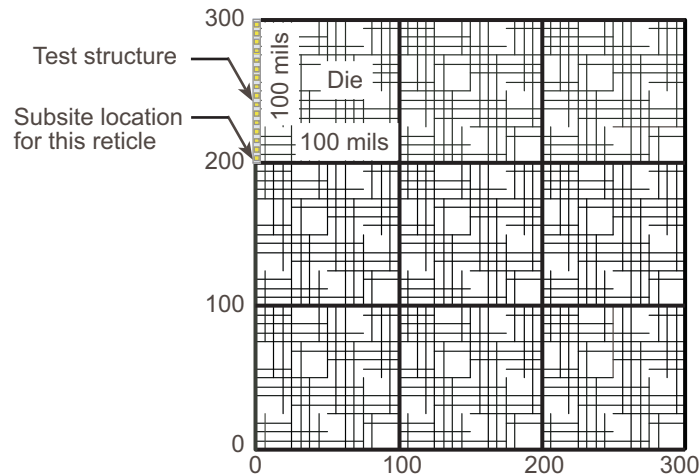
*Figure 2-11*  
**Subsite coordinates 0,200 for a site that is defined as a single 300 mil x 300 mil die**



- 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 [Figure 2-12](#).

Figure 2-12

**Subsite coordinates 0,200 for a site defined as a reticle with nine 100 mil x 100 mil dies**



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 will be prompted to create the file or cancel.

## Control buttons

There are six control buttons along the top of the Site Editor window. The function of each button is as follows:

**Add Site Aft.** — Used for multi-project 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 multi-project 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.

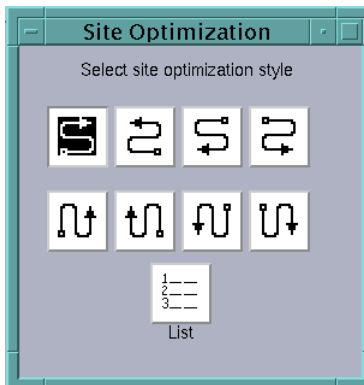
**Delete** — Deletes the currently selected field.

**Copy Rows** — Will copy the rows indicated in the Copy Rows window after the row where the cursor is currently positioned.

## Site Optimization window

The Site Optimization window, [Figure 2-13](#), is used to specify the order in which each of the test sites on the wafer are tested.

*Figure 2-13*  
**Site Optimization window**



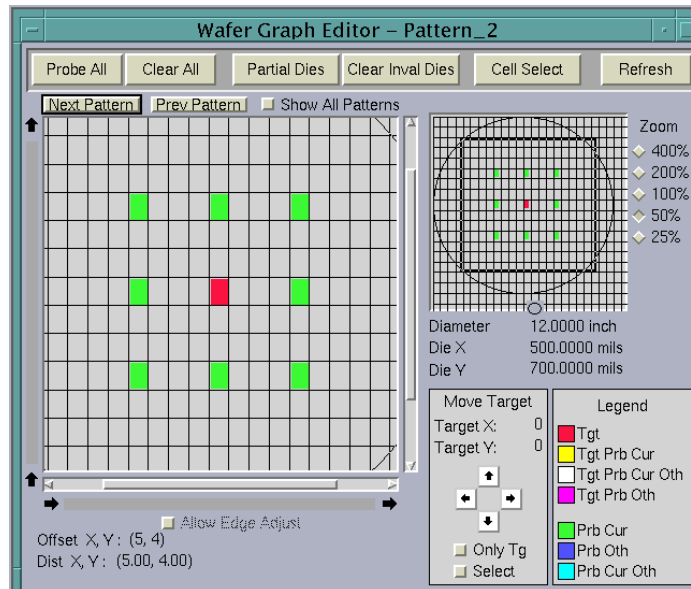
You may choose from eight different patterns and the List selection. When a pattern is selected, the tester starts at the site closest to the chosen location and then proceeds to all selected sites on the wafer, following the selected serpentine pattern. When the List selection is chosen, the system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

## Wafer Graph Editor window

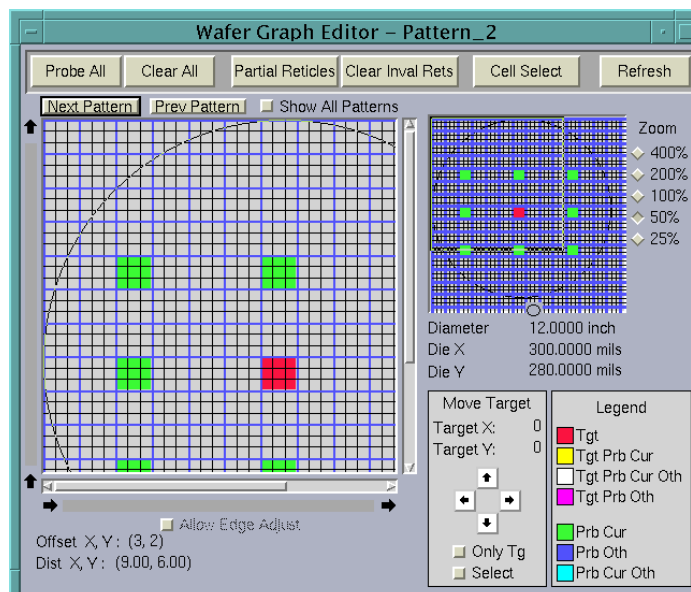
The Wafer Graph Editor window, [Figure 2-14](#) and [Figure 2-15](#), works with single-project .wdf files only. This window displays a single probe pattern at a time or all probe patterns contained in a .wdf file (see the Show All Patterns button above the wafer display). The left mouse button can be used to select additional site locations to be included into a probe pattern.



**Figure 2-14**  
**Wafer Graph Editor when “Sites are Dies” is selected in the Wafer Setup window**



**Figure 2-15**  
**Wafer Graph Editor when “Sites are Reticles” is selected in the Wafer Setup window**



Note in [Figure 2-15](#) that the graphical site display at left shows both the full reticles as well as 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.

- If you choose Partial Dies, you may then select dies that do not fall completely within the usable wafer space.
- If you choose Complete Dies, you may not 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.) Moreover, if any previously selected dies straddle usable wafer boundaries, such dies will be 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.

- If you choose Partial Reticles, you may then select reticles that do not fall completely within the usable wafer space.
- If you choose Complete Reticles, you may not select reticles that do not fall completely within the usable wafer space. (For example, a Probe All action will then select only reticles that are fully within the usable wafer space.) Moreover, if any previously selected reticles straddle usable wafer boundaries, such reticles will be 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.

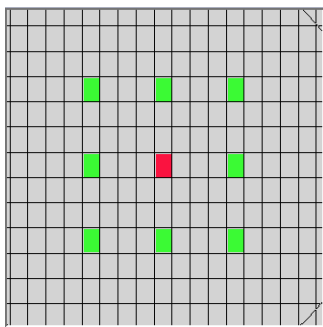
**Clear Inval Rets** — (Visible only when Sites are Reticles is selected in the Wafer Setup window.) When Complete Rets has been previously selected, clears all the reticles selected for testing that do not fall completely within the usable wafer space.

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

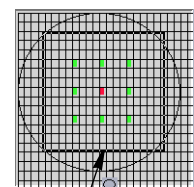
**Refresh** — Refreshes the Wafer Graph Editor window with all changes or updates made to the data from all the other windows.

## Graphical editing features

### Graphical site displays



The specific sites represented in the large pattern at left correspond to the sites inside the selection frame of the small pattern at right. In turn, the size of the selection frame is determined by the Zoom settings, as discussed below in “[Zoom settings](#).” The selection rectangle may be moved with the mouse to any location on the wafer. Therefore, using a combination of Zoom settings and selection rectangle positions, the sites on any part of the wafer can be viewed



Selection frame

and edited at various magnifications.

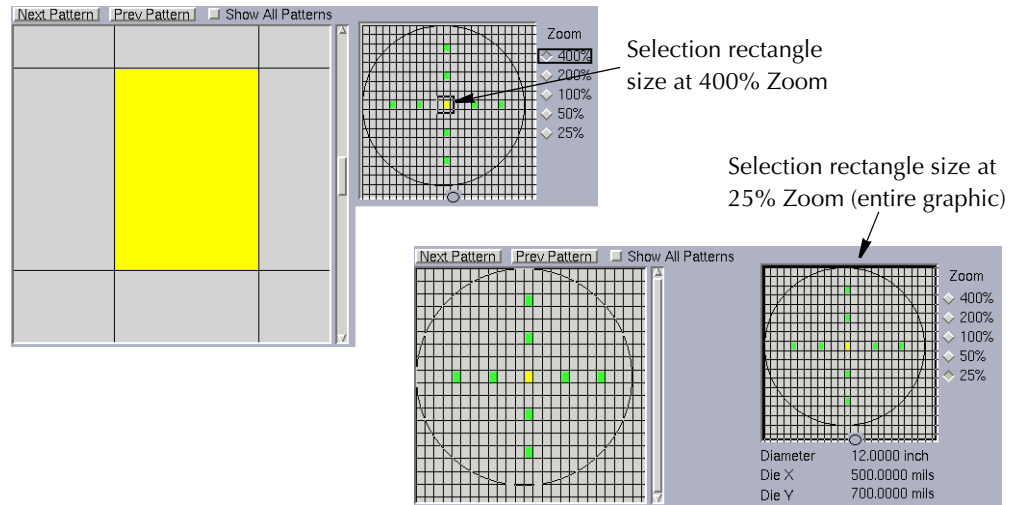
Target sites in the large pattern on the left may be directly selected or deselected by clicking. Target sites may be manipulated by the arrows and selections at the lower

right of the Wafer Graph Editor. Refer to [“Move target arrow buttons and location display”](#) on page 2-20.

### Zoom settings

The Zoom settings at the far right control the size of the selection frame (see figure above) and the magnification of the sites displayed in the large graphical pattern. See [Figure 2-16](#).

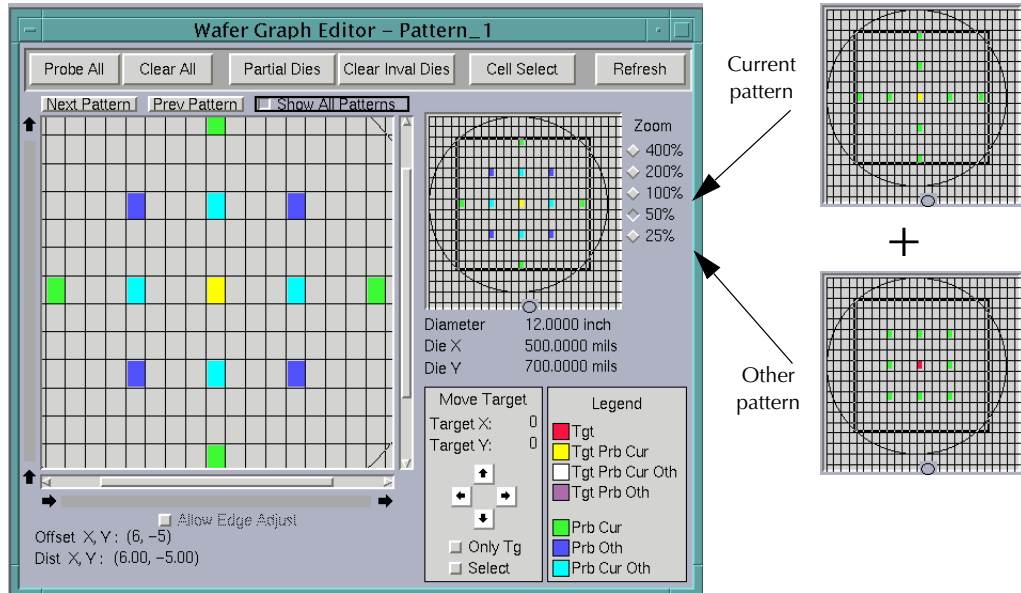
*Figure 2-16*  
**Using Zooms**



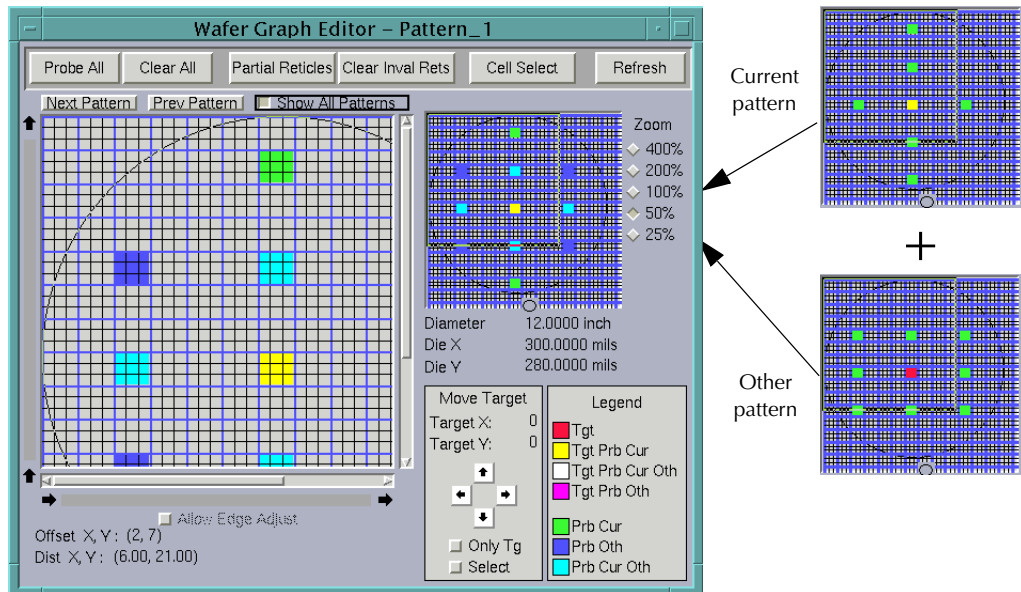
### Controls located above the graphical site displays

- **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. Figures [2-17](#) and [2-18](#) illustrate the Wafer Graph Editor when Show All Patterns is selected.

**Figure 2-17**  
**Effects of selecting Show All Patterns for a Sites are Dies .wdf (compare Figure 2-14)**



**Figure 2-18**  
**Effects of selecting Show All Patterns for a Sites are Reticles .wdf (compare Figure 2-15)**



Note that color codes distinguish the sites in the current probe pattern from the sites in other probe patterns. The color codes in Figures 2-17 and 2-18 show that the other (non-current) 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.

## Legend

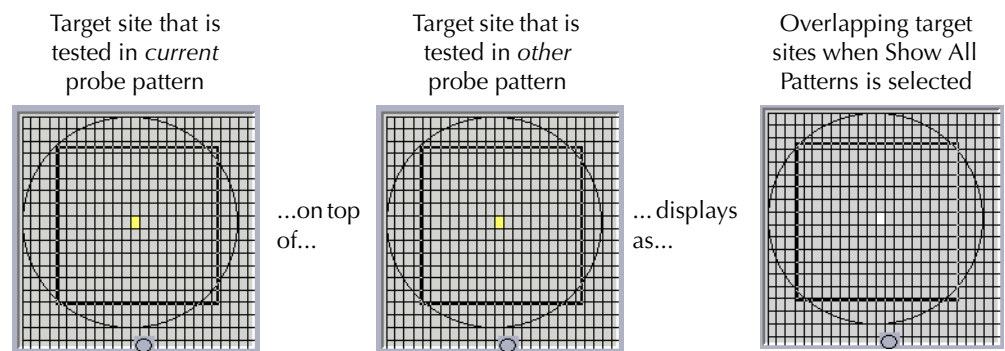
The site colors in the graphical site displays are coded as shown in the legend. This subsection explains the meaning of each color, as follows:

- **Tgt (red)** — Target-only site (not tested).
- **Tgt Prb Cur (yellow)** — Target site that is tested in the current probe pattern.
- **Tgt Prob 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 [Figure 2-19](#).

Legend	
<span style="color: red;">■</span>	Tgt
<span style="color: yellow;">■</span>	Tgt Prb Cur
<span style="background-color: white; border: 1px solid black;">■</span>	Tgt Prb Cur Oth
<span style="color: magenta;">■</span>	Tgt Prb Oth
<span style="color: green;">■</span>	Prb Cur
<span style="color: blue;">■</span>	Prb Oth
<span style="color: cyan;">■</span>	Prb Cur Oth

Figure 2-19

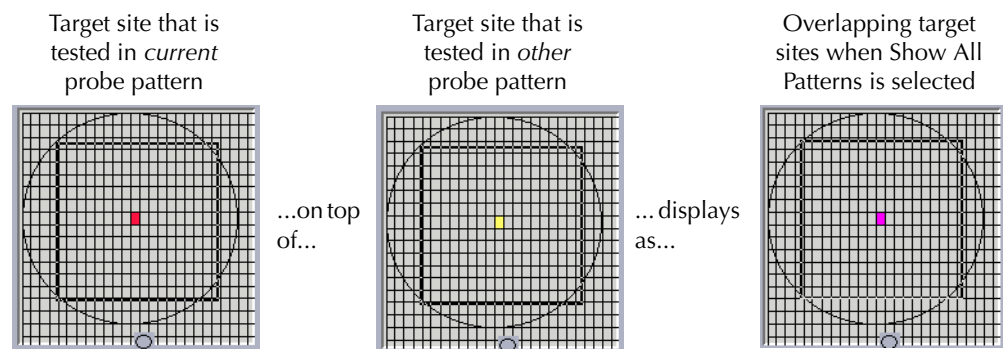
### Scenario that results in Tgt Prob Cur Oth (white) color-coded target site



- **Tgt Prob 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 [Figure 2-20](#).

Figure 2-20

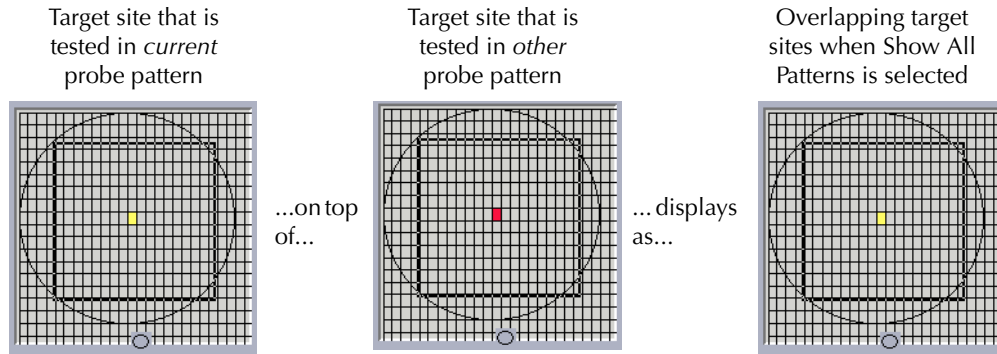
### Scenario that results in Tgt Prob Oth (magenta) color-coded target site



## NOTE

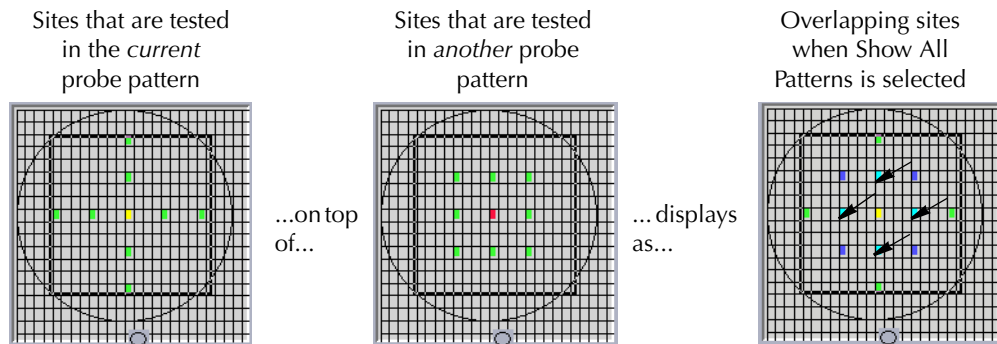
However, note that when a tested target site in the current probe pattern overlaps an untested target site(s) in another probe pattern, the target site color is yellow. See [Figure 2-21](#).

Figure 2-21

**Tested target site in the current probe pattern overlapping another untested target site**

- **Prb Cur (green)** — Probed (non-target) site that is tested in the current probe pattern.
- **Prb Oth (blue)** — Probed (non-target) site that is tested in another probe pattern. Applicable only when Show All Patterns is selected.
- **Prb Cur Oth (cyan)** — Probed (non-target) 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 [Figure 2-22](#).

Figure 2-22

**Example of overlapping probed sites in current and non-current probe patterns****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.

**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 three ways.

- **Move the target site by clicking the arrows —**
  1. First select the Only Tgt button, which allows only the target site to move.
  2. Then, click on an arrow button to move the target site one position in a given direction.
- **Move the target site by selecting a destination —**
  1. First, select both the Only Tgt and Select buttons.
  2. Then, select a new target site location by clicking it on the graphical display. The target site, only, moves to the new location.
- **Move all selected sites by clicking the arrows —**
  1. First, select the Select button.
  2. Then, click on 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 since all the sites on the wafer are the same.

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

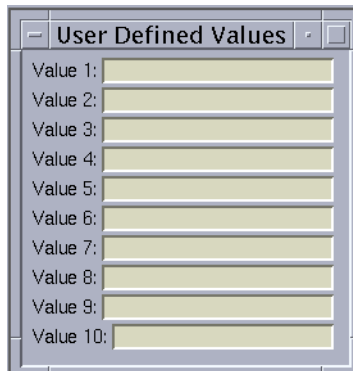
### **X and Y offset fields**

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**

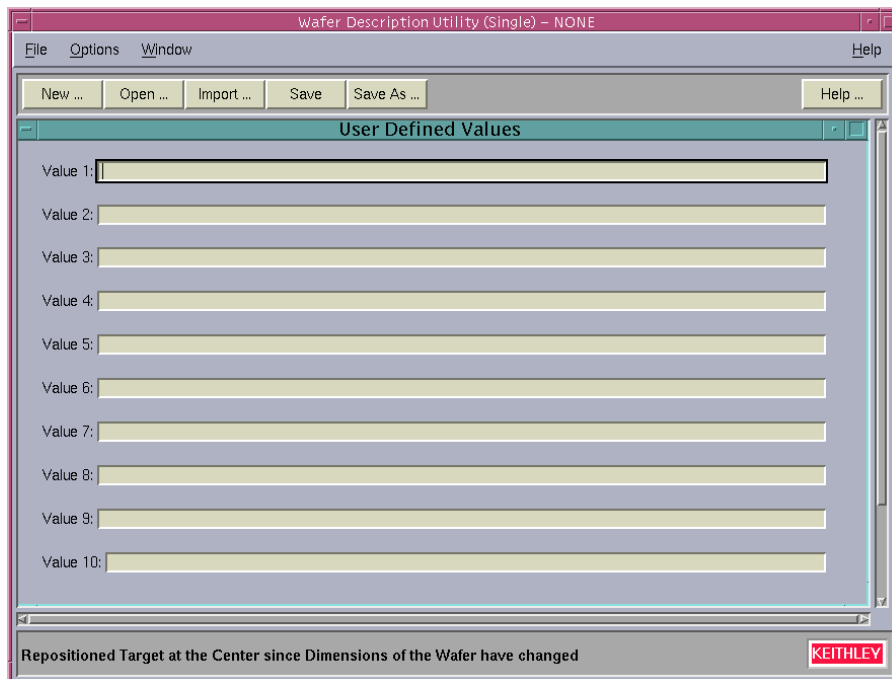
The User Defined Values window contains ten fields. See Figures [2-23](#) and [2-24](#).

**Figure 2-23**  
**User Defined Values window, as it first appears**



**NOTE** If this window is not visible among the layers of WDU windows, bring it forward by 1) clicking the middle mouse button and 2) selecting **User Defined Values** from the menu that appears.

**Figure 2-24**  
**User Defined Values window, expanded**



In these fields, optionally enter information that you would like to be available to KTXE — for example, as parameters for UAP code that you write. The information must meet the following requirements:

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

The WDFRec pointer in the datapool will contain these user-defined values. This pointer can be used in UAP routines after KTXE loads the Wafer Description file.





## 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 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 on 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 in any of the cells.

## Copying device data

Device data may be cut, copied, and pasted within the same instance of TSE, or between separate instances. To cut or copy device data, select the device data and click the right mouse button. A Device Actions pop-up menu will appear. Select Cut or Copy to place the device data on the clipboard. Cut will remove the device data from the current location and place it on the clipboard. Copy will leave the device data in its current location and only place a copy of it on the clipboard.

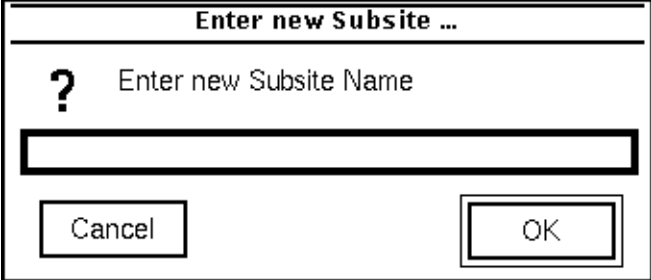
Select a new location in either the same instance of TSE, or in another instance, and right click again. The Device Actions pop-up menu will appear. Select Paste Before or Paste After and the device data will be inserted in the selected location.

## Creating a test structure file

To create a test structure file, perform the following steps:

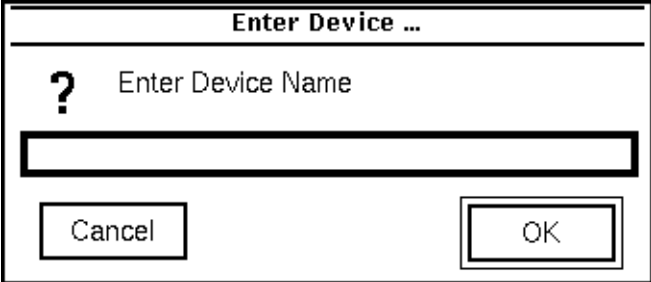
1. Select New from the File menu. A text box, [Figure 2-26](#), will appear asking you for the Subsite name. Enter the name of a subsite that is currently in use in a wafer description file (.wdf) and click OK.

*Figure 2-26*  
**Subsite name test box**



2. Next a text box, [Figure 2-27](#), will appear asking for a device name. Enter the name of the device and click OK. To add additional devices to the current list, click on 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 2-27*  
**Device name text box**



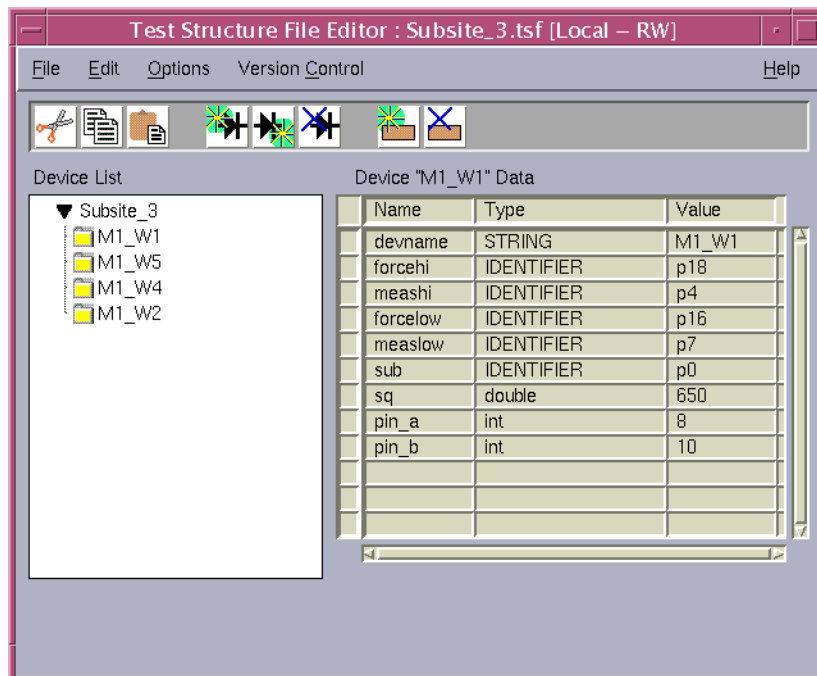
3. To enter device data, select the device from the Device List, then select the Add Info After icon or menu item. The first row will already contain the default parameter name, new\_name. Enter the parameter name to be used.

- In the Type column, click in the cell to activate the pop-up menu. Click on the arrow to the right of the cell to open the menu, and select the proper parameter type. The parameter type, IDENTIFIER, references pre-defined probe card or global identifier data.

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

- In the Value column, enter the actual value for the parameter.
- 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, see [Figure 2-28](#), and can be accessed by the KITT Parameter Entry window if that subsite is selected in the KITT subsite box.

**Figure 2-28**  
**Completed test structure file**



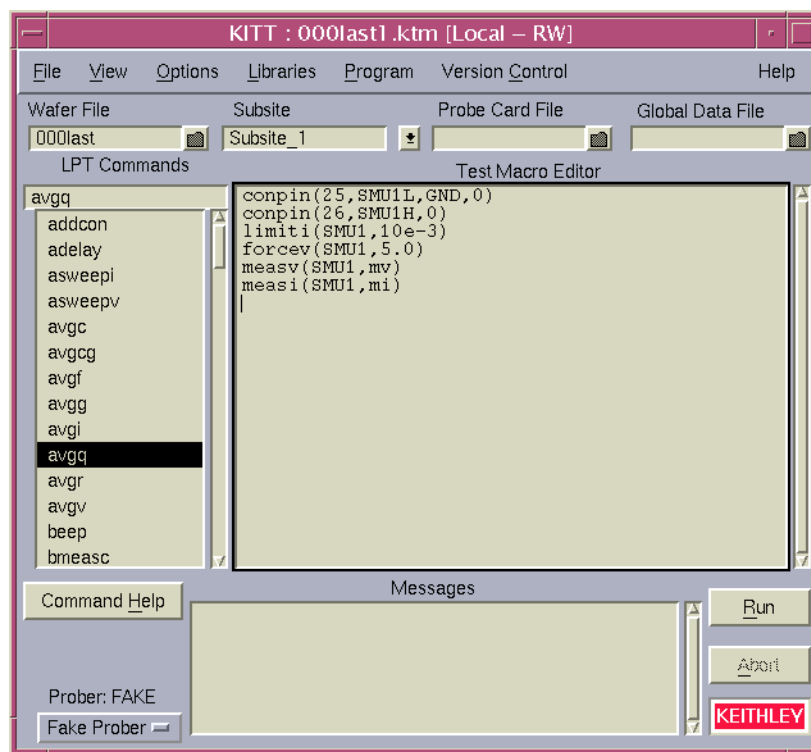
## Keithley Interactive Test Tool (KITT)

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

Figure 2-29 shows the KITT main window that appears on the screen when KITT is first activated. A brief description of each feature follows.

Figure 2-29  
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 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 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 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 option menu controls:

- Whether the tester hardware is online or offline.
- If auto-plotting is on and 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 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
- LPT

- PAR
- User Libraries

### KITT program menu

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

This information is written into the macro file (.ktm) so future work in 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 figure out 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 on 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 may be moved from the Parameter Entry window to the Test Macro Editor window by 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.

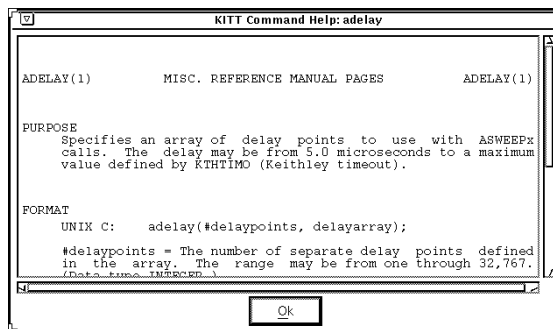
## Messages window

The Messages window displays event and error messages that occur during 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 [Figure 2-30](#).

*Figure 2-30*  
**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 on 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

[Figure 2-31](#) shows the 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 2-31  
**KITT results window**

KITT Results						
File		Options				
0	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						
7						
8						

## KITT Math, Array and Logic Expression Support

### Math

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

#### Definition

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

- Pre-defined 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 (i.e. 1.23e-4)

The operators/functions above may also be used with the equal sign (=) assignment operator. This operator can assign a new value to previously defined identifiers or create 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 will occur if a divide by 0 is attempted. The macro execution will only terminate 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

### Definition

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

- pre-defined identifiers within a test macro file (.ktm)
- global data from a global data file (.gdf)

A complete array or a single element may 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 may be used to define the array index.

If a variable is to be used as an array index, this variable MUST BE type INT. This is necessary for the “Practice Task” and “Save As C” features to function correctly.

## Examples

### 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` 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 )
```

### 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()
```

### 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( parm1, 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 V3.2 and up.

### 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 V3.2 and up.

### if-then-else and Logical Expressions

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 will generate a 0 or 1 response. If a result is created and assigned using a Boolean operator, the result type will be 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 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 identifier are of three 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 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 KITT Parameter Entry window will place the parameter set data identifiers in all symbolic value fields that have a matching parameter. If a parameters 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 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-z or A-Z. This is set in the kitt.ini file.

## KTE data usage

### Datapool

- A data storage area generated during KTXE and KITT execution.
- Run-time storage of Probe Card File (.pcf) data, Global Data File (.gdf) data, Global PDI data, KTXE control data, and data generated with dpAdd functions. Each run will start with a clean datapool.

### Data precedence

KTXE and KITT resolve parameters from the data sources in the following order:

- Local PDI
- Test Structure Data
- Parameter Set Data
- Probe Card File Data
- Global Data File Data
- Global PDI
- Generated Results

### Data sources

#### Local predefined identifiers (PDI)

- A Local PDI is a value local to a KTM.
- A Local PDI is highest in the data precedence. A Local PDI value will be used before the datapool.
- 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/device specific data.
- Changing test structure data will effect 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 will permit modification of all KTMs that use the parameter set.

#### Probe card file data (PCF)

- Probe Card File data is loaded into the datapool during the wafer loading phase of KTXE.
- Since the PCF file data is loaded for each new wafer plan, items from previous PCF files may be in the datapool.
- KITT will load 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 (GDF)

- Global Data File data is loaded into the datapool during cassette plan loading phase of KTXE.
- .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 will be used.
- KITT will load the .gdf file at the start of each run.
- Useful for passing data between KTM.
- 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.
- UAPs can access this data.

### Global predefined identifiers (PDI)

- A Global PDI value is available to the defining KTM. After KTM execution, they are placed into the datapool. Any KTM executed following the defining KTM or re-execution of the defining KTM will get the value of the Global PDI from the datapool.
- A Global PDI is lowest in the data precedence.
- Useful for passing data between KTM. (Global data is more flexible.)

### Generated results

- Output variables generated by KTXE or KITT.

### KTXE control data

- KTXE places items in the datapool at run-time. This data is used internally by KTXE and may be altered to change KTXE execution behavior.
- See Appendix D, Datapool, for a list of these items.
- During KITT execution, no KTXE Control Data exists.

## Passing data between KTM

Data is passed between KTM through the datapool. KTM may place data into the datapool by outputting the results to a GDF defined name or a Global PDI, or using a `dpAdd` function in a user library. Datapool items accessed by a KTM must exist before executing a macro line. If not, a run-time error will occur and the KTM currently executing will terminate.

Example:

Two KTM each generate one result that is used by a third KTM. Two values will be placed in the global data file that is loaded in the datapool.

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,
    slope_5x10 )
```

KTM — calc.ktm

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

## Use of ibupu in KITT

In 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 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 allows the user to add, remove, or edit parameter set data assigned to a library module. The PSE does not modify the default data created in 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 KITT parameter entry screen. If the Parameter Set Editor is active when this save occurs, the Parameter Set Editor will not be refreshed with the new data until the desired Module name is re-opened. Clicking on the folder icon for the Module will refresh the display.

### PSE main window

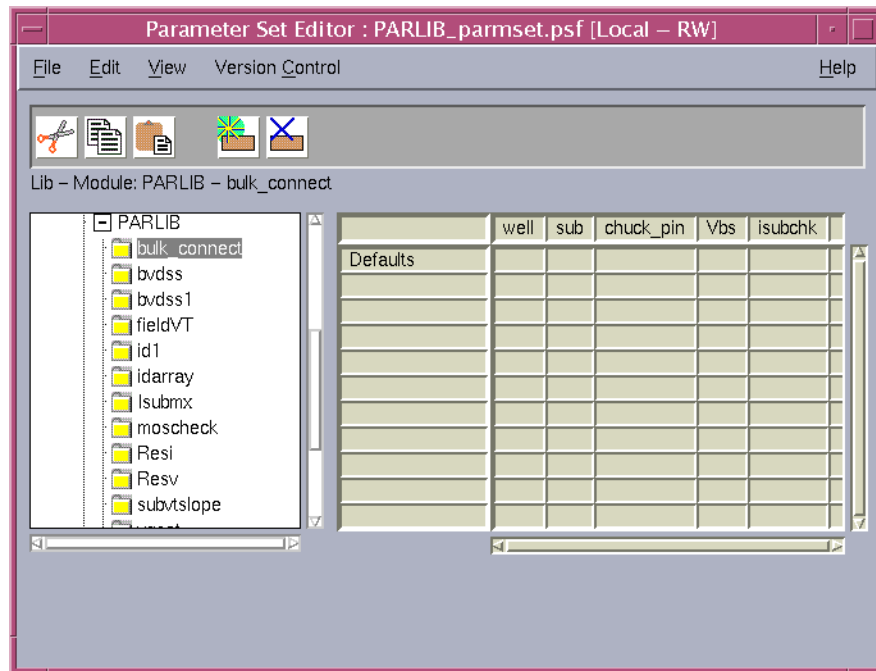
There are two main areas to PSE, shown in [Figure 2-32](#):

**The Library Browser** — Allows the user to navigate through a directory tree of available user libraries and modules to select the desired parameter data.

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



Figure 2-32  
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 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 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 PSE help documentation.
- **About** — Displays the software revision level.

## Library browser

To navigate through the available libraries, click on the plus (+) icons to expand the library tree and the minus (-) icon 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 on the folder icon opens the parameter data contained in the selected module into the Parameter Data list box.

If the directory tree is longer than the viewable window, a vertical scroll bar will appear that can be used to scroll the library/module list into view. Holding the left-mouse button down and dragging on an empty section of the directory tree background will also allow 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 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 alpha-numeric character or an underscore character. The name of the parameter set is made available in the Parameter Set Selection box in the 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 KTXE.

This field will display 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 in 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 on 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 will be 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 on the parameter set name cell and press the Delete Parameter Set button. You will be asked to confirm the deleting. 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/or 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 will be purged from the data structures. The user must save this parameter set in order to completely remove the data from future use.

## Keithley User Library Tool (KULT)

KULT is a tool used to create and manage user modules and libraries. It performs a different function than the other 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, the user will edit and compile C 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. Thus, 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 will be compatible with future software upgrades.

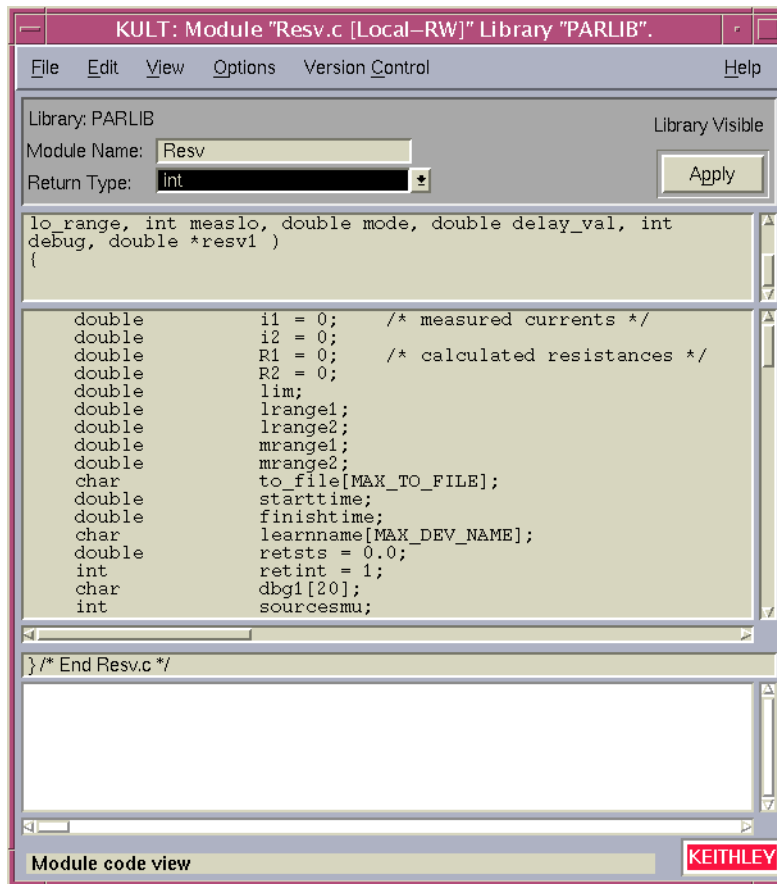
KULT libraries are created to allow for easy and quick extensions to the capabilities of the parameter test routines (PAR), prober drivers, and the production test execution engine (KTXE). Once a "user library" is developed, it is available for test macro development in KITT. Or, the module can be included 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 KULT main window is shown in [Figure 2-33](#). 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 2-33  
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.

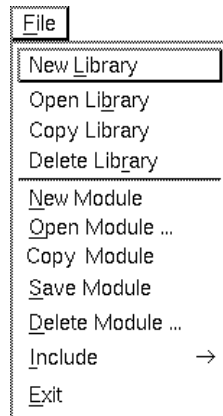
### KULT file menu

The File menu shown in [Figure 2-34](#) 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 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 KULT.

Figure 2-34  
**KULT file menu**



### KULT edit menu

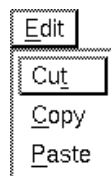
The Edit menu shown in [Figure 2-35](#) contains file editing functions.

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 2-35  
**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:

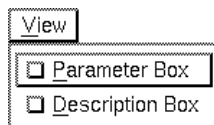
- **Cut** — Same as Cut in Edit menu.
- **Copy** — Same as Copy in Edit menu.
- **Paste** — Same as Paste in Edit menu.

- **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 [Figure 2-36](#) for the View menu options.

*Figure 2-36*  
**KULT view menu**

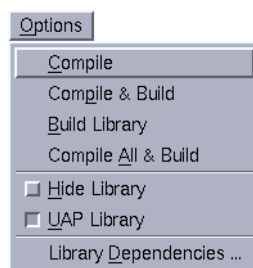


### KULT options menu

The Options menu shown in [Figure 2-37](#) 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 KITT, between visible and hidden.
- **UAP Library** — Toggles the availability of the presently open library for use at a UAP.
- **Library Dependencies** — Allows the user to specify other user libraries to link to.

*Figure 2-37*  
**KULT options menu**



### KULT help menu

The Help menu contains online help information about 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 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 KITT and/or available to UAPs in KTPM. For details, refer to [“Changing library attributes” on page 2-52](#).

## 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 lets you move through the code.

## Error/warning message area

This area will contain any error and/or warning messages generated by the C compiler. Selecting (double-clicking) a compile error listed in this window will move 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 will appear: “No error/warnings reported, compilation/build was successful.”

## Parameter window

The Parameter window shown in [Figure 2-38](#) is located at the top right side of the 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 2-38  
Parameter window

KULT Parameters: Module "NoName" Library "TEST"					
Parameter Name	Data Type	I/O	Default	Min	Max
pin_a	int	Input	25		
pin_b	int	Input	27		
pin_c	int	Input	29		
sq	double	Input	325	300	350

Include Files

```
#include <stdio.h>
#include <lptdef.h>
#include <lptdef_lowercase.h>
#include <math.h>
```

## 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 on 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 on the arrow to the right of the I/O field will activate a pop-up menu that shows the input and output selections.



### Default, min, and max fields

These three fields are used to specify the following:

- Default field — For input parameter: sets the default value for the specified parameter. For output parameters: 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 parameters, 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

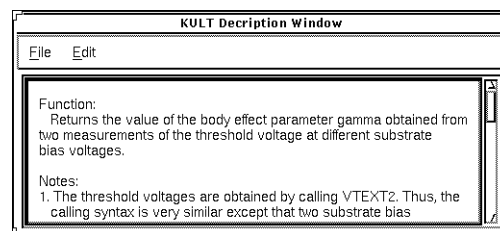
The control button area is aligned vertically along the right side of the parameter box and is composed of the following functions:

- **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 will 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 [Figure 2-39](#) is located at the bottom of the KULT screen. Information entered in this window is for module documentation and KITT user library help. For more information on KITT, refer to the Keithley Interactive Test Tool section.

*Figure 2-39*  
**Description window**

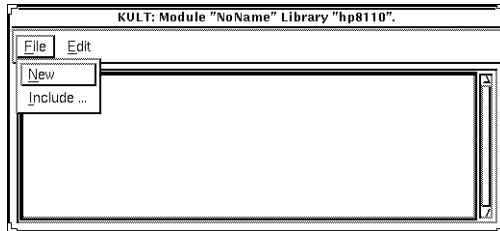


### File menu

The File menu shown in [Figure 2-40](#) 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 2-40  
Description window file menu



## Edit menu

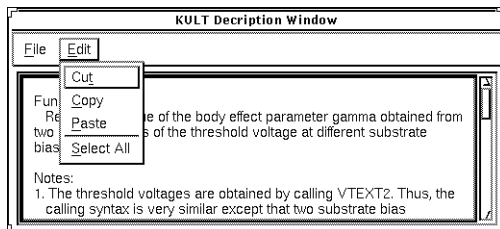
The Edit menu shown in [Figure 2-41](#) contains the following selections:

- **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 2-41  
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 KITT Command Help.*

## Command line interface

The 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 include:

- new\_lib
- bld\_lib
- del\_lib
- add\_mod
- del\_mod
- compile\_mod
- update\_mod

The options include:

- -l<library\_name>
- -d<directory\_name>
- -hide
- -nohide
- -uap
- -nouap
- -dep [library\_name].....[library\_name]

### **new\_lib**

The new\_lib subcommand lets you start KULT and immediately load a new library from the command line. The command is:

```
kult new_lib -l<library_name>
```

### **bld\_lib**

The bld\_lib subcommand lets you start creating a new library immediately after starting KULT from the command line. The command is:

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

### **del\_lib**

The del\_lib subcommand lets you delete a library from the command line. The command is:

```
kult del_lib -l<library_name>
```

### **add\_mod**

The add\_mod subcommand lets you add a module immediately after starting KULT from the command line. The command is:

```
kult add_mod -l<library_name> -d<directory_name> <module>
```

The `-d` option is needed when the modules you want to add are not in the present directory.

### **del\_mod**

The `del_mod` subcommand lets you delete a module from the command line. The command is:

```
kult del_mod -l<library_name> <module>
```

### **compile\_mod**

The `compile_mod` subcommand lets you compile a module into the library `Library_name`. The module must already exist. The command is:

```
kult compile_mod -l<library_name> <module>
```

### **update\_mod**

The `update_mod` subcommand 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 command is:

```
kult update_mod -l<library_name> <module>
```

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 wish to add a `usrlib` module from another facility or project to the current project. Please note that 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.

### **Example**

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

If you wish to add the module to an existing library, simply 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]]
```

`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 KTE installation packages to the current KTE installation package when the current package is installed in an alternate directory. In fact, it can be used to copy any usrlib from any project of any previously installed package into the current `KI_KULT_PATH` directory.

`migrate_usrlib` first looks for previously installed packages, then prompts the user 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 the user 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/project, and prompts the user to select one (or all) to copy. It proceeds to call `kult_copy_lib` to copy the selected usrlib(s) into the `KI_KULT_PATH` directory, and then exits.

Usage:

```
migrate_usrlib [-p]
```

## Locking a module

Each time a module is opened within KULT, a lockfile 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 will appear stating that the module selected is locked and will give the present user's name and PID.

After clicking on the OK button, the module will open 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 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 lockfile is created at \$KI\_KULT\_PATH/"libname"/lock in the following manner:

The lockfile is named X - Y .lck, where X is the library name and Y is the module name.

The following information is contained within 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/Lockfilename.

## Changing library attributes

### Hiding libraries from KITT macros

A library that you create may contain modules that will not work if called by KITT macros — for example, modules that are designed only to be called by other modules in other user libraries. The “Hide Library” menu item, as well as the analogous - hide command-line option, allows you to make such a library unavailable to macros by hiding the library from KITT.

#### Application example

Suppose 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 KITT, use either of the following methods.
  - Using the KULT window, do the following:
    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.
  - Using the command line, enter the following:
 

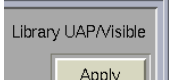
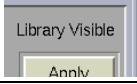
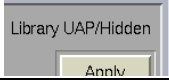
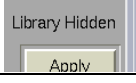
```
kult bld_lib -l<library_name> -hide
```
- To make a user library visible in KITT, use either of the following methods.
  - Using the KULT window, do the following:
    1. Open the library in KULT.

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.
- Using the command line, enter the following:  
kult bld\_lib -l<library\_name> -nohide

**Table 2-1** summarizes “Hide Library” use combined with “UAP Library” use (discussed next).

*Table 2-1*

**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 <sup>2</sup>	Options menu selections <sup>2</sup>	Available to test macros and visible in KITT	Available at UAPs and in KTPM <sup>3</sup>	
-nohide <sup>1</sup> -uap <sup>1</sup>	<input type="checkbox"/> Hide Library <input type="checkbox"/> UAP Library	●	●	
-nohide <sup>1</sup> -nouap	<input type="checkbox"/> Hide Library <input type="checkbox"/> UAP Library	●	No	
-hide -uap <sup>1</sup>	<input type="checkbox"/> Hide Library <input type="checkbox"/> UAP Library	No	●	
-hide -nouap	<input type="checkbox"/> Hide Library <input type="checkbox"/> UAP Library	No	No	

1. Default state.

2. These commands or Options menu selections change the present state of the library by overriding and changing values in the `settings.ini` file.

3. Modules used at KUAP locations must be compiled with the `-uap bld_lib` command option.

## Setting library availability at UAPs

Some libraries that you create may contain modules that do not work in UAPs. The “UAP Library” toggle, when deselected, as well as the analogous `-nouap` command-line option, make such libraries unavailable at UAPs and in KTPM. Refer to the procedures below and [Table 2-1](#).

- To make a user library unavailable at UAPs and in KTPM, use either of the following methods:
  - Using the KULT window, do the following:
    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.
  - Using the KULT window, do the following:
    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

### Run-time library locking

#### LIBRARY reader locks for KTXE, KSOX, and KITT

These locks will be 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/tools use the Library reader locks:

#### KSOX

When we 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 will be created since the `dlopen` was not called. Delete the reader lock when a library is closed.

#### KITT

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

KITT will use 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

#### LIBRARY write lock for KULT

This lock will be created and exist only while 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 will check for the presence of any reader locks before attempting a “build library” operation. If a read lock exists, a warning message will be shown to the user and the library build will not occur.

KULT will create a write lock before the build operation starts. This write lock will be 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 will not be 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, will leave reader locks present in the \$KI\_KULT\_PATH/lock directory.
- Using the “kill -9” command:  
If a process is stopped via the “kill -9” command, reader locks will not be removed automatically from the \$KI\_KULT\_PATH/lock directory.

Simply deleting the lock files will resolve the above problem(s).

## Limits File Editor (LFE)

LFE creates and edits limits files (.kif). 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

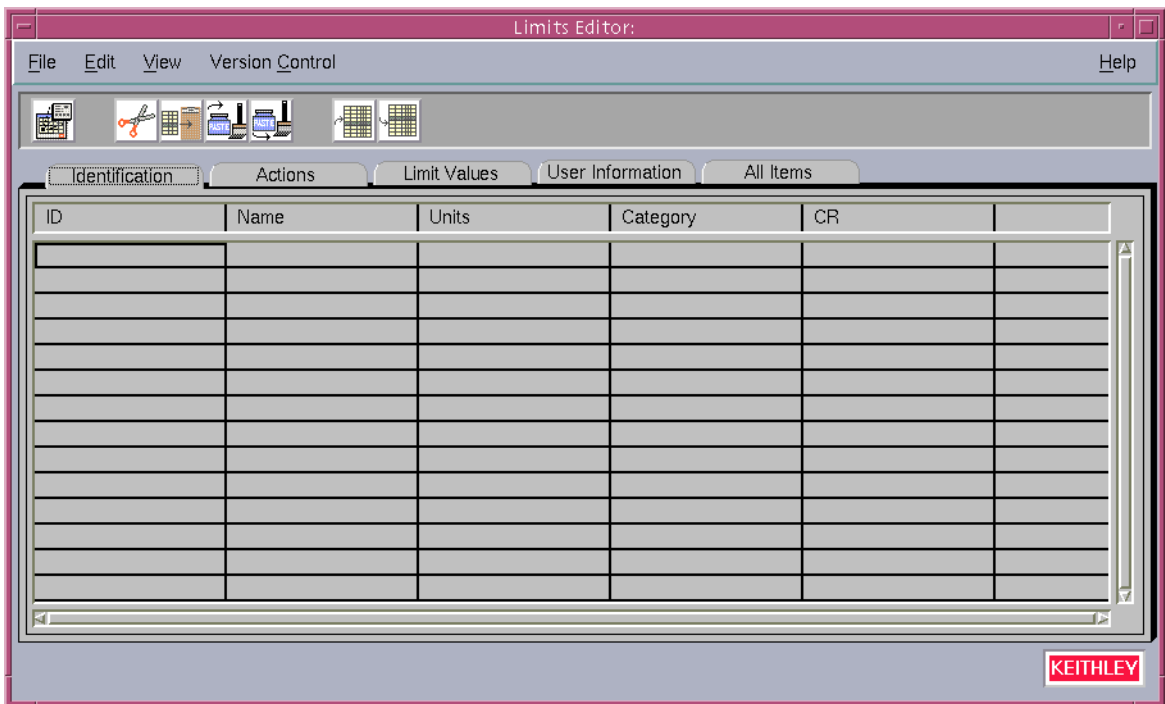
Valid, Spec, Ctrl, and Engr limit pairs. The limits file used during production testing is specified in the Wafer Plan.

## LFE main window

The LFE Main window shown in [Figure 2-42](#) is the primary interface of LFE.

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

*Figure 2-42*  
**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 pulldown menus contain the functions to select and save parameter limits files; to exit LFE; to manipulate data in a limits file; to view limits file information in a different format; and to access help. The following paragraphs contain a brief overview of each menu.

### LFE file menu

You can use the File menu to reinitialize the main window (new), to load or save limits files, to delete limit files, to generate limits from KTM (Keithley Test Macros) files, to set a limit as the default limit, to save column settings, and to exit 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 LFE and the release version of the LFE being run.

## Pushbuttons

Seven pushbuttons 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 the user to quickly find and modify the desired fields. There are five tabs:

- **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” name. The string may contain a maximum of 128 characters. This field is required.

### Name

The name field provides for a “friendly” name, perhaps describing 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” name. The only constraint is that it must not include commas. The string may 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” name. The

only constraint is that it must not include commas. The string may contain a maximum of 10 characters. Examples of entries for this field are volts, amps, etc.

### **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, as long as all the categories are listed (comma- or whitespace-delimited). This field may contain a maximum of 20 characters.

### **RP**

RP (Report) determines whether or not the limit is included in the lot summary report compiled in 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 will loop 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 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 are invalid usually due to an equipment malfunction.
- **Spec** — This limit pair 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 will be 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 may contain a maximum of 255 characters.

### User fields

The User fields may 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 [Figure 2-43](#). 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.

This window also tracks the movement of the cursor in the 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 the user to select one of the available options for each field.

Figure 2-43  
Limits editor dialog window

## Keithley Test Plan Manager (KTPM)

The following paragraphs contain basic information on 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.

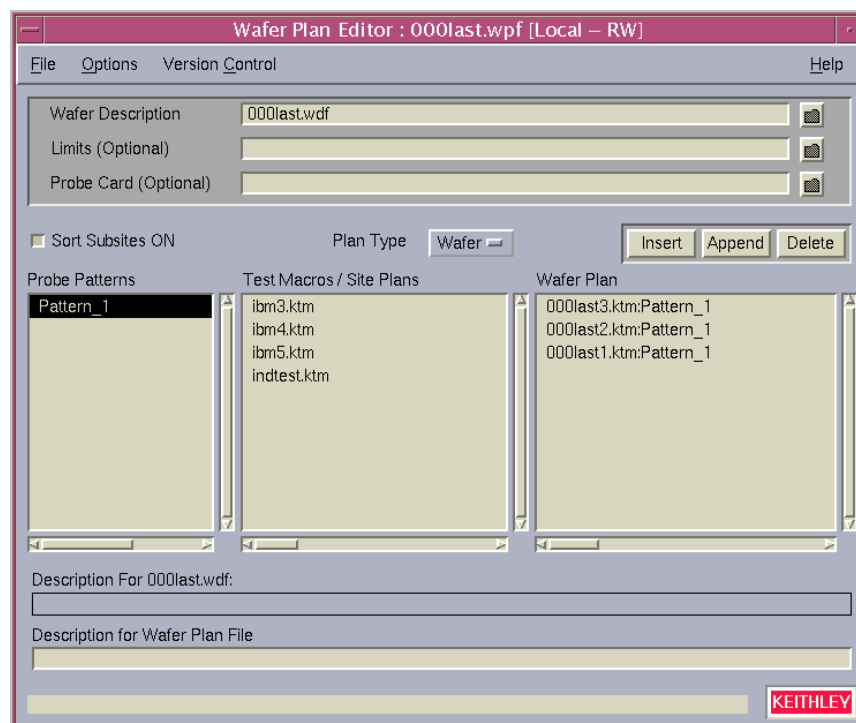
### Wafer plan builder main window

The Wafer Plan Builder window shown in [Figure 2-44](#) 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 2-44  
**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/disables the storing of the various filenames within the .wpcf file. The default environment variables will be 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 will always be saved in the .wpcf file.

## Wafer plan builder help menu

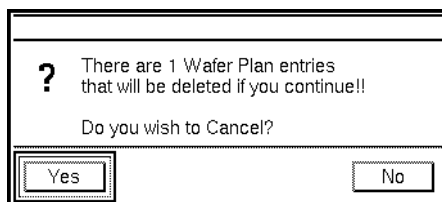
The Help menu provides on-line help and release version information about the Wafer Plan Builder. The Documentation option allows access to the on-line KTPM manual. The About option displays the software revision information.

## Wafer description file selection field

The wafer description file selection field lets you select the .wdf file that will be used during test execution of this wafer plan. The .wdf file selected will be read to display the defined probe patterns. Right-clicking on the filename in this field displays a pop-up dialog allowing you to open the wafer file for editing.

If a test macro/site plan has been entered into the wafer plan and the .wdf folder button is selected, a warning box shown in [Figure 2-45](#) will appear 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 2-45*  
**Wafer plan entry warning box**



## Limits file selection field

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



## Probe card file selection field

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

## Plan type selection button

Click and hold on this button and there will be a choice between building a Wafer Plan and building a Site Plan.

## Sort subsites button

When this button is selected, the subsites will be 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/site plans list

When the plan type selection button is set to Wafer Plan, this area will contain a list of all the probe patterns available from the wafer description file selected. When Site Plan is selected, this area will list 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 .wpfs.

## Test macros/site plans list

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

## Site plan/wafer plan builder field

This field is used to build the wafer plan or site plan. By clicking on the different files in the Probe Pattern/Site Plan list and the Test Macros/Site Plan list, and then clicking on the Insert Bef. or Insert Aft. buttons, you will 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/site plans are removed from the list of available macros/site plans. Each macro/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 functions of each of these buttons is:

- **Insert button** — Will insert 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** — Will insert 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** — Will remove 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 on 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 .wfp file whenever the site plan is used in that .wfp. This new site plan will appear 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

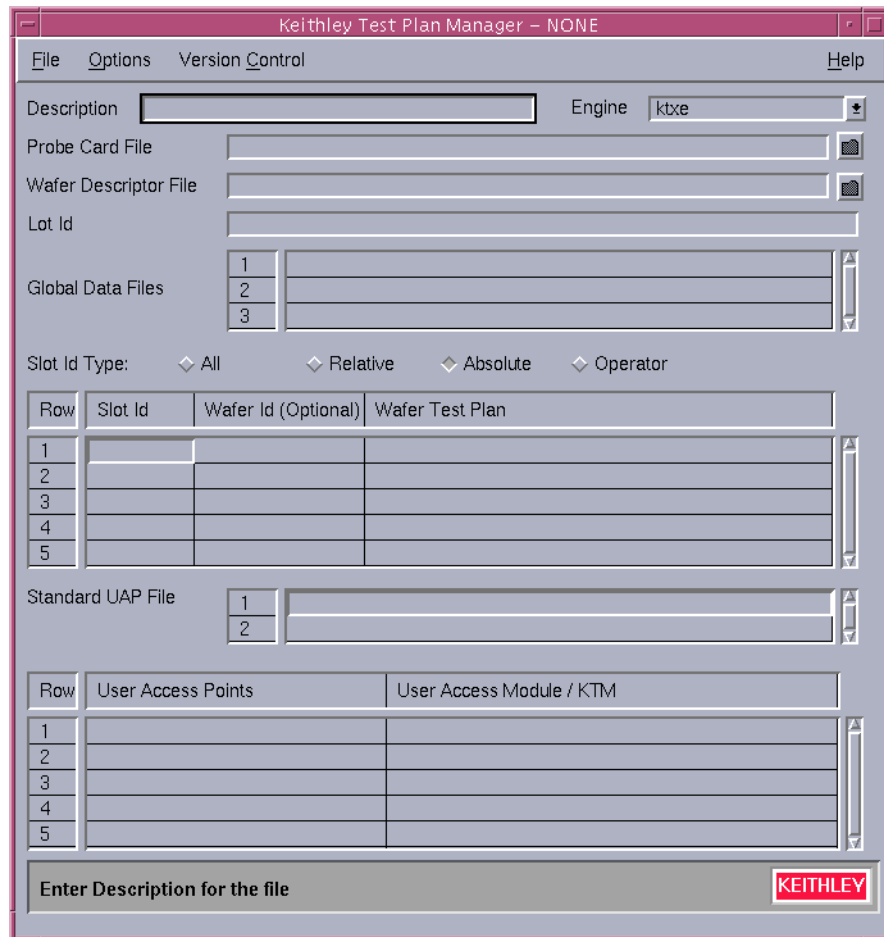
Displays the description of the selected .wdf file. This field cannot be edited.

# Cassette plan builder

The Cassette Plan Builder shown in [Figure 2-46](#) contains the following areas:

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

Figure 2-46  
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 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/disables the storing of the various filenames within the .cpf file. The default environment variables will be 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 will always be saved in the .cpf file.

## Cassette plan builder help menu

The Help menu provides on-line help and release version information about KTPM. The Documentation option allows access to the on-line 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, if entered, will override the .pcf file specified in any wafer plan file used. Right-clicking on the filename in this field displays a pop-up dialog 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 that, if entered, will override the .wdf file specified in any wafer plan file used. Right-clicking on the filename in this field displays a pop-up dialog allowing you to open the wafer description file for editing.

## Lot ID data field

This is an optional field that lets you identify the lot that will be tested. This data will be 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 be 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 UAP file field

The Standard UAP file field is optional and lets you specify files that contain module and KTM routines to be used at UAPs in the execution engine.

## UAP module/KTM area

The UAP Module/KTM area lets you specify the module or .ktm file that will be executed at the 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 will appear 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 KTPM. The tool provides a report that shows each UAP and .ktm file that is executed, in the order of execution for all wafers, sites, and subsites.

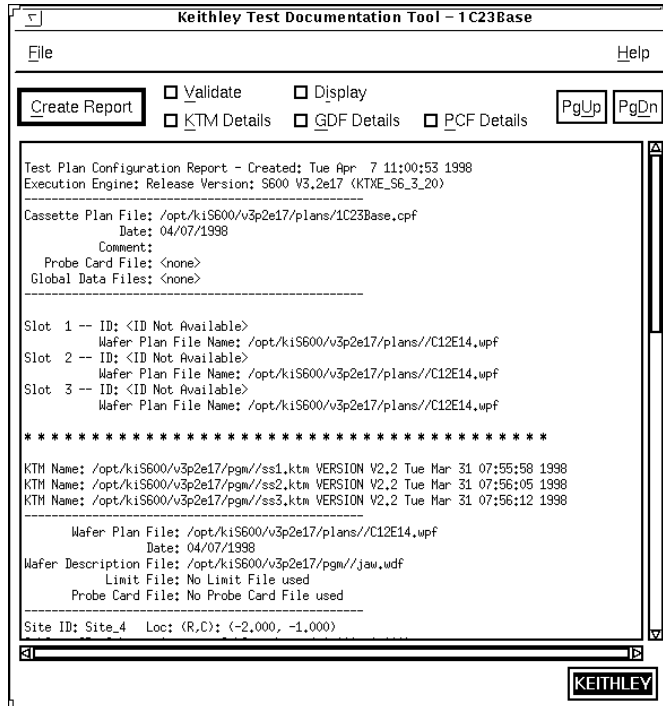
## Main window

The Test Documentation Tool main window, shown in [Figure 2-47](#), contains the following areas:

- **Main menu bar** — Contains the File and Help menus.
- **Create report button** — Will regenerate the report with any new options specified.
- **KTM details toggle** — Will enable or disable the display of KTM details.
- **GDF details toggle** — Will enable or disable the display of Global Data File details.
- **PCF details toggle** — Will enable or disable the display of Probe Card File details.
- **Validate toggle** — Will enable or disable additional cassette plan validation steps.
- **Display toggle** — Will enable or disable 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 2-47  
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 on-line help and release version information about the Test Documentation Tool. The Documentation option allows access to the on-line KTDT manual. The About option displays the software revision information.

## Create report button

When this button is clicked, the report will be created with either the KTM, GDF, and PCF Details enabled or disabled.

### **KTM details toggle**

This toggle button is used to enable or disable the display of all 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.

### **GDF details toggle**

This toggle button is used to enable or disable the display of the Global Data File 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.

### **PCF details toggle**

This toggle button is used to enable or disable the display of the Probe Card File 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 pathnames 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/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**

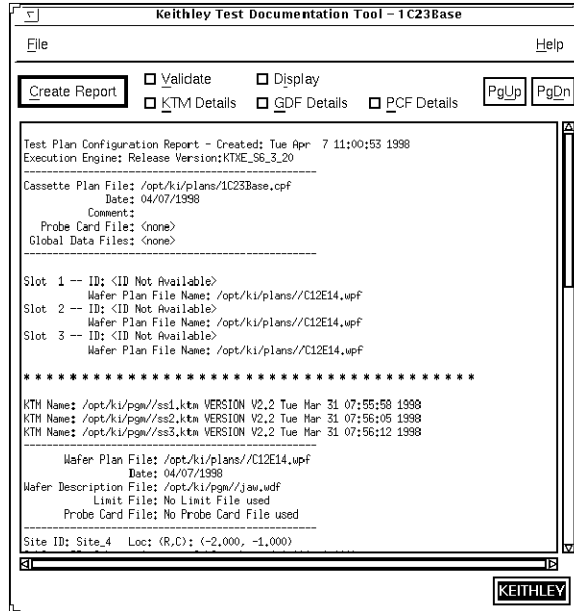
This field is where the entire test report can be viewed. If a wafer plan file has been specified for multiple slots, only the full details of the first occurrence of the wafer plan file will be shown.

## **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 will appear as shown in [Figure 2-48](#).

**Figure 2-48**  
**Test documentation tool**





Enable/disable the appropriate options for your report and press the Create Report button. The system will create and display 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 desired tests in the proper sequence. For all wafers, sites, and subsites, you can also examine each of the test macros by clicking on the 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 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 will be 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

### Overview

Adaptive testing is an optional component of 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 pre-defined zones, or patterns, contained in the wafer definition file. For information on creating patterns in the wafer definition file, refer to *The Wafer Description Utility (WDU)*.

### Zone based test modes

Zone based testing can be initiated by two different methods. The first method, described below, is a command line switch for use with KTXE. The second method,

described under Global Data Variables, 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 desired. For example:

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

The four test modes are as follows:

- **N – Normal Test** — When the 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 will be retested using (pattern number + 1) of the original so 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 datapool 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 UAPs. The following routines are included in the `KTXE_RP` user library:

- **KTXE\_RP\_CleanUpWDF** — This function renames the randomly generated WDF file to `lot_XXX_wdfname`. This preserves the WDF file used in the test for a future adaptive retest. This routine is used if a lot execution is suspended.  
This function should be called at `UAP_LOT_END`.
- **KTXE\_RP\_CreateRandomWDF** — This function creates a wafer file based on the maximum frequency and number of wafers. The name of the wafer description file is saved in the `cpf_info` structure. This function uses the `rand_pat` utility to generate random patterns. For information on the `rand_pat` utility, refer to *KTE Support Utilities*.  
This function should be called at `UAP_WRITE_LOT_INFO`.
- **KTXE\_RP\_CreateWPF** — This function creates a new wafer plan file to use the appropriate probe pattern name. The new wafer plan file is created by duplicating the base file and modifying the probe pattern name.  
This function should be called at `UAP_WAFER_PREPARE`.
- **KTXE\_RP\_GetUsrArgs** — This function checks the command line arguments used to invoke KTXE to determine the test mode. This overrides the global data entry, `KTXE_RP_test_mode`, if used.  
This function should be called at `UAP_PROG_ARGS`.
- **KTXE\_RP\_RemoveWPF** — This function deletes wafer patterns that were generated at run time.  
This function should be called at `UAP_WAFER_END`.

## Required UAP files

A UAP file, `adapt_zone.uap`, has been provided to illustrate the usage of these library commands. This UAP file must be specified in 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 desired UAP file. Care should be taken in using the UAPs. Execution order is important. Refer to Appendix E for more information on UAPs and Section 5, *Environment variables*, 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 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 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 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	Runtime
2	A	Additional Sites
4	S	Runtime 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/or 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.

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 WDU using alternate test patterns, and have a 1-to-1 relationship with the normal sites. Normal testing resumes on the next site.
- **More Sites on Current Wafer** — When testing on any site fails, additional pre-defined sites are tested. There is a 1-to-many relationship between the normal sites and the alternate sites. Normal testing does not resume on the next wafer, 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, as long as 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 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 (LFE)*.
- **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 - 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 UAP and GDF files also need to be specified.

For information on using the Limits File Editor, refer to *Limits File Editor (LFE)*. For information on using the Wafer Description Utility, refer to *The Wafer Description Utility (WDU)*.

The following paragraphs describe the requirements for each of the four 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 desired tests.

1. Start KTPM and open the CPF file that will be adapted to use result based testing.
2. Add the required UAP file that corresponds to the desired 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 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 GDF file that corresponds to the desired test mode. The specific files required are listed in the description of the individual test modes. Set the global data variables as required by the desired testing mode.
6. Set up the limits file using 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 desired 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 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 KTPM. The only exception is that the appropriate GDF and 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 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 KTPM. The only exception is that the appropriate GDF and 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 for all wafers in the lot, as long as no new wafer plan is encountered.

The first step in implementing this test mode is to define the critical parameters using 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 LFE.

After all the component files have been set up, the test is set up as normal with KTPM. The only exception is that the appropriate GDF and 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 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 KTPM. The only exception is that the appropriate GDF and 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\_next\_wafer.gdf and adapt\_more\_tests\_next\_wafer.uap.

**KTXE\_AT user library**

Result based testing is accomplished with the use of calls to the user library KTXE\_AT at various UAPs. The following routines are included in the KTXE\_AT user library:

**KTXE\_AT\_alternate\_site\_site\_end**

Format `KTXE_AT_alternate_site_site_end()`  
 Description This function is used to determine, at completion of site testing, if an alternate site should be tested.  
 Use at `UAP_SITE_END`

**KTXE\_AT\_alternate\_site\_test\_end**

Format `KTXE_AT_alternate_site_test_end()`  
 Description This function is used to determine if any test results failed during the testing of a ktm.  
 Use at `UAP_TEST_END`

**KTXE\_AT\_AlterWWP**

Format `int KTXE_AT_AlterWWP(long *current_wwp_list, float altx, alty)`  
 Description This function is used to alter the wwp list to add all ktm's that have failed on the current site to be executed at an alternate site. This function is used internally.

**KTXE\_AT\_CheckResWithLimits**

Format `int KTXE_AT_CheckResWithLimits(char *Result_Name, double Result_Value, int Limit_Code, char *Limit_List)`  
 Description This routine should be called from another routine or from a UAP to check the value of a result with the limits specified in a limit list or a limit sublist. This function is used internally.  
 Parameters **Result\_Name** — Character string specifying the name of the result. The name should exist in the limit list.  
**Result\_Value** — Double value of the result.  
**Limit\_Code** — An integer value between 1 and 4. The result is checked against Valid, Spec, Ctrl or Engr limit based on the code. 1 = check against "Valid" limit. 2 = check against "Spec" limit. 3 = check against "Ctrl" limits. 4 = check against Engr limits.  
**Limit\_List** — The name of the limit list to use for looking up the result. You can pass the default "limit\_list" or pass in a limit sublist name for this parameter.  
 Return value If the result is within the limits TRUE is returned; FALSE is returned otherwise.  
 Run at UAP This routine can be run at any UAP as long as the Result and the limit list exist in the datapool at that UAP. This routine can also be called from other routines (Example: KI\_SubsiteTest).  
 Helpful hints
 

- This routine returns a TRUE or a FALSE. The return value should be checked after this routine is called and action should be taken based on the returned value.
- To see the errors generated by this macro when running in KTXE, set the `KI_KTXE_ERROR_LOG` environment variable.

**KTXE\_AT\_cleanup\_site**

Format `KTXE_AT_cleanup_site()`



**Description** This function is used to reset counts and clean up data structures. This function is used internally.

### **KTXE\_AT\_debug\_print**

**Format** `KTXE_AT_debug_print()`

**Description** This function is used to print the internal flags and counters used in KTXE\_AT.

**Use at** UAP\_SITE\_END

### **KTXE\_AT\_demo\_data\_func**

**Format** `double KTXE_AT_demo_data_func(double x)`

**Description** This function will generate demo data. It is called internally by KTXE\_AT\_generate\_val.

### **KTXE\_AT\_enable\_kdf**

**Format** `KTXE_AT_enable_kdf()`

**Description** KTXE\_AT\_enable\_kdf is used to enable kdf logging if disabled earlier.

**Use at** UAP\_TEST\_DATA\_LOG

### **KTXE\_AT\_FindAltSite**

**Format** `KTXE_AT_FindAltSite()`

**Description** This function is used to find the alternate site coordinates for the current site. This function is used internally.

### **KTXE\_AT\_generate\_val**

**Format** `double KTXE_AT_generate_val(int demo_type, double highV, double lowV)`

**Description** This demo data generation function returns a value for each variable tested based on highV and lowV, which are upper and lower bounds, and the demo\_type:

0random high/low fail  
 1inside fails high  
 2outside fails high  
 3top fails high  
 4bottom fails high  
 5right fails high  
 6left fails high  
 7inside fails low  
 8outside fails low  
 9top fails low  
 10bottom fails low  
 11right fails low  
 12left fails low

Call this function from a ktm as follows:

```
result = KTXE_AT_generate_val( demo_type, demo_high_lim,
demo_low_lim )
```

result — (double) a test results

demo\_type — (int) a number 0 - 12 which defines result failures based on site location on wafer

demo\_high\_lim — (double) upper bound of passing results

demo\_low\_lim — (double) lower bound of passing results

Define demo\_type, demo\_high\_lim, and demo\_low\_lim as global data for ease of use.

**KTXE\_AT\_LogResultList**

Format            KTXE\_AT\_LogResultList(char \*result\_list\_name)  
Description      This function is used to log saved result lists to kdf. The function is used internally.

**KTXE\_AT\_more\_sites\_cur\_wafer\_site\_end**

Format            KTXE\_AT\_more\_sites\_cur\_wafer\_site\_end()  
Description      This function is used to determine, at completion of site testing, if more sites should be tested on the current wafer.  
Use at            UAP\_SITE\_END

**KTXE\_AT\_more\_tests\_curr\_wafer\_site\_end**

Format            KTXE\_AT\_more\_sites\_cur\_wafer\_site\_end()  
Description      This function is used to determine, at completion of site testing, if more tests should be executed on the current wafer.  
Use at            UAP\_SITE\_END

**KTXE\_AT\_more\_tests\_curr\_wafer\_wafer\_begin**

Format            KTXE\_AT\_more\_tests\_cur\_wafer\_wafer\_begin()  
Description      This function is used to add results in the alternate test class to the list of results that are disabled during initial execution.  
Use at            UAP\_WAFER\_BEGIN

**KTXE\_AT\_more\_tests\_next\_wafer\_site\_end**

Format            KTXE\_AT\_more\_tests\_next\_wafer\_site\_end()  
Description      This function is used to change wafer plan to be used for the next wafer that has the same wafer plan name as the current wafer plan.  
Use at            UAP\_SITE\_END

**KTXE\_AT\_wafer\_begin**

Format            KTXE\_AT\_wafer\_begin()  
Description      Library initialization module.  
Use at            UAP\_WAFER\_BEGIN

**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\_".

# KTE Support Utilities

## Random Pattern Generation (rand\_pat)

The `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` options are 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 re-testing. 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 will be uniformly achieved.

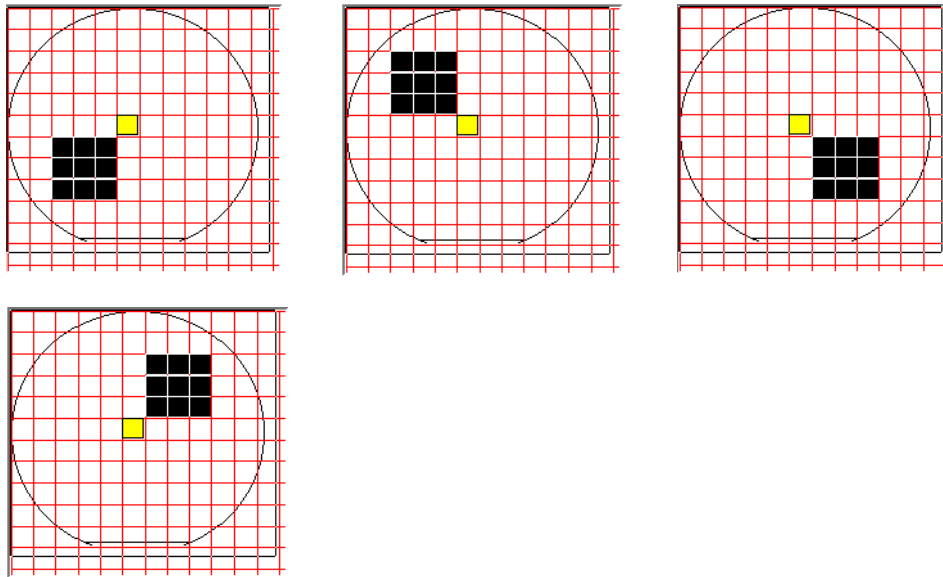
At this point, the utility allows you to specify a number of sites per wafer different from the number of patterns. It will also allow you to generate a number of patterns different from the *natural* number. If fewer patterns are specified, there may be untested sites; on the other hand, if more are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

Example:

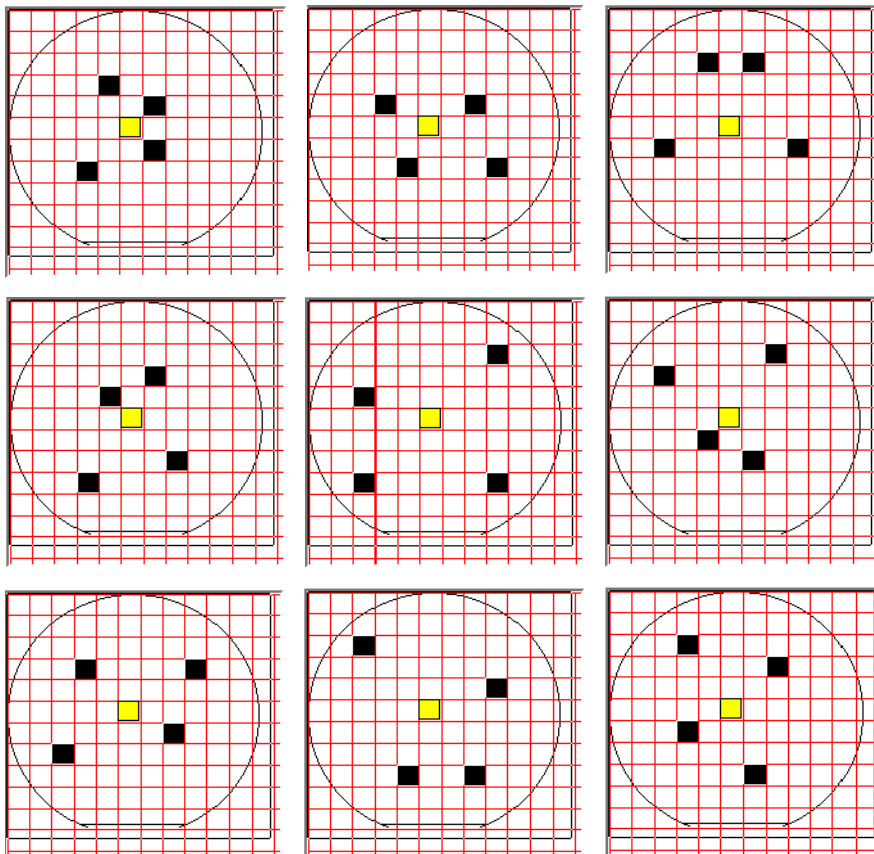
Upon executing

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

where the input file zones have the following form:



The output patterns look like:



## File filtering

All 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 box 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 pre-defined 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`. 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.

# 3

## Test Execution

---

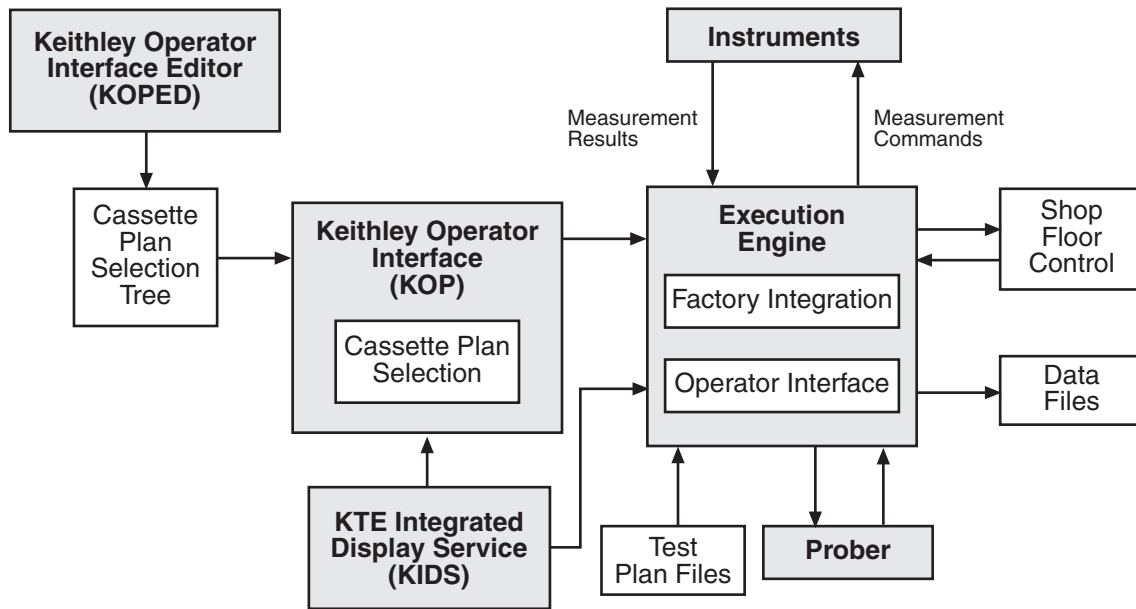
## Introduction

You can execute a cassette plan one of four ways:

- Using KIDS
- Using KOPED and KOP
- Directly accessing KTXE

Figure 3-1 shows a diagram of the test execution process.

Figure 3-1  
Test plan execution flow diagram



This section covers the tools required to execute a completed test plan. In this section the following will be discussed:

- **Keithley Operator Interface Editor (KOPED)** — This section will cover how to create a series of test processes that can be activated from the Keithley Operator Interface (KOP).
- **Keithley Test Execution Engine (KTXE)** — This section will cover how to use KTXE to execute the test plans created using KTPM.
- **KTE Integrated Display Service (KIDS)** — This section will cover KIDS operation for recipe selection, execution, and display status.

When you have completed this section, you will be able to utilize KSU and KCAT, covered in Section 4, to analyze your test results and generate a report.

## Keithley Operator Interface Editor (KOPED)

This section includes basic information for using KOPED. Brief descriptions of KOPED's main window and menus are also included.

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 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 the user wishes.

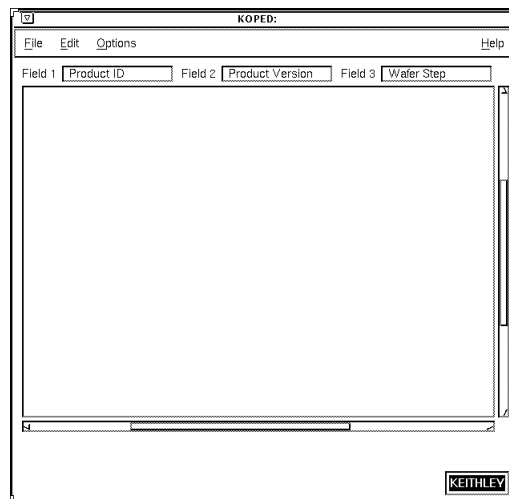
## KOPED main window

The main window shown in [Figure 3-2](#) 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 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 3-2*  
**KOPED main window**



## KOPED file menu

The 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 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

This menu lets you enable or disable Test Plan Validation. If enabled, the operator will be able to use the Verify button on the 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 Help menu provides on-line help and KOPED version information. The Documentation option allows access to on-line 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 desired 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 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)

This section includes basic information on the Keithley Test Execution Engine (KTXE). A brief description of KTXE's main window is included.

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 KTE KUI localization in Appendix B.

The KTXE interface can be viewed in any language. Refer to Appendix F, *KTE Localization*.

## KTXE main window

The KTXE main window shown in [Figure 3-3](#) 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 on-line help for KTXE.

Figure 3-3  
Lot dialog window

Lot Information - System: s600q4023 TS: 1

Operator: kthmgr40

Lot Id: arrTest

Process:

Device:

Test Name: ktxe arrTest.cpf

Limit Id:

System Id: s600q4023 Test Station: 1

Search 1:

Search 2:

Search 3:

Comment:

KEITHLEY Help OK ABORT

## Test execution from KTXE

It is also possible to execute your cassette plan by directly accessing KTXE.

1. Start KTXE from the command line by entering the following: `ktxe -cpf filename.cpf` where `filename.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 [Figure 3-4](#).
3. Click on OK to begin the testing process.

Figure 3-4  
Lot dialog window

The screenshot shows a dialog box titled "Lot Information - System: s600q4023 TS: 1". The fields are as follows:

- Operator: kthmgr40
- Lot Id: arrTest
- Process: (empty)
- Device: (empty)
- Test Name: ktxe arrTest.cpf
- Limit Id: (empty)
- System Id: s600q4023
- Test Station: 1
- Search 1: (empty)
- Search 2: (empty)
- Search 3: (empty)
- Comment: (empty text area)

At the bottom left is the KEITHLEY logo. At the bottom right are buttons for Help, OK, and ABORT.

### Command-line options for KTXE

The following is a listing of all the command-line options that can be entered when starting KTXE.

- c text lot information comment field  
This switch can be used to add comment text to the lot header.
- cass n select cassette n: 1-4  
This switch is used to select a specific cassette for a prober with multiple cassettes.
- cpf fname 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 lot information device field  
This switch can be used to specify the device field of the lot header.
- doc "options" Documentation tool info  
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 KTM execution
NO_UAP	Disable UAP execution

DOC\_ON Enable Test Plan Document data generation. This will cause KTXE to create a datafile containing execution information. This datafile is named casPlanName.cpf.kxd and will be placed in the directory specified by the \$KITMP environment variable.

DOC\_ONLY Configure for Test Plan Doc Gen 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".

- e n [fname] error reporting mode n: 0 - 3
  - 0 - None
  - 1 - Display messages
  - 2 - Log messages
  - 3 - Display and Log messages
- ev n [fname] event reporting mode n: 0 - 3  
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 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 lot search key n: 1 - 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 uses specified recipe name  
See the Keithley Recipe Manager documentation for more information.
- l id 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 file(s).
- o text lot information operator field  
This switch allows the operator field in the .kdf lot file to be filled with the specified text.
- p text lot information process field  
This switch allows the process field in the .kdf lot file to be filled with the specified text.
- r "options" lot summary report options  
This switch allows the sum\_report\_options datapool 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 lot information system field  
This switch allows the system field in the .kdf lot file to be filled with the specified text.
- u text user argument  
This switch can be used for additional user command-line arguments.
- w fname 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 debug flag n: 0 - 32767  
This switch can be used for user command-line arguments.

fname — valid filename and path

options — options must be enclosed in quotes

text — command line arguments will be concatenated to the switch argument until the next switch (dash-letters) occurs

id — valid file name, no paths or extensions

n — numeric

## The execution process

Once test execution has begun, the execution engine begins working its way through the cassette plan you specified. 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. [Figure 3-5](#) shows a flow diagram of how the execution engine proceeds through all of the test data.

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 KTM level where the test execution process begins. The engine finds each specified site and subsite 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 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 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 (KTPM).
- Subsites will be probed in the order they are specified in the Site Editor (WDU).

**Figure 3-5**  
**Test execution engine**

Cassette Level	Wafer Level	KTM Level
process the command line arguments load cassette plan UAP_PROG_ARGS get lot id UAP_CASSETTE_LOAD UAP_LOT_INFO display lot dialog screen initialize prober for this cassette UAP_PROBER_INIT prompt operator to load cassette UAP_WAFER_MISMATCH proper pipeline and cassette plan verified UAP_ACCESS_WDF_INFO access WDU before call to PrInit UAP_POST_PROBER_INIT change prober setup based on WDU file UAP_WRITE_LOT_INFO write lot structure into kdf file UAP_POST_LOT_INFO load wafer UAP_WAFERLOAD_STATUS UAP_PROFILE_WAFER provides time for hot chuck to warm up wafer before profiling. UAP_ALIGN_ERROR UAP_POST_INITIAL_WAFER_LOAD perform AutoZ after first wafer load	UAP_WAFER_PREPARE load the wafer plan load the probe card file load the limits file load wafer description file prepare the complete wafer plan load all the ktm's UAP_VALIDATE_OCR write wafer header to lot file UAP_WAFER_BEGIN	move probe chuck to the test site UAP_SITE_CHANGE write site data to lot file move probe chuck to the test subsite UAP_SUBSITE_CHANGE UAP_TEST_BEGIN execute the ktm UAP_TEST_END write results to lot file UAP_TEST_DATA_LOG check limits file/results for abort UAP_HANDLE_ABORT UAP_SUBSITE_END UAP_SITE_END
UAP_WAFER_END write end of wafer to lot file load next wafer UAP_ALIGN_ERROR write end of lot to lot file UAP_LOT_END UAP_ENGINE_EXIT		
UAP_ABORT_EXIT_HDLR } UAP_PRB_ERR_HDLR } The UAPs to the left are executed at the time the event happens. UAP_STATUS_CHANGE }		

The following is an in-depth description of what happens at each 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 datapool 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 DataPool. The GetDocumentationConfig routine will adjust the values if necessary.
- Load cassette plan.
- Load Global Data files only IF there are files to load.
- Refresh “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 UAP before displaying lot dialog screen to operator.

**UAP\_LOT\_INFO**

Display Lot dialog screen and gather information from Operator.

Display status dialog on screen.

Starting Tester Control.

Add the LOT ID to the data pool.

The product filename should be set at this 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 since product file could enable/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 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 datapool in case a UAP modified the value.

This datapool value can be set for probers that need the first wafer manually loaded.

( P8 and others )

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\_ALIGN\_ERROR**

Error recovery after wafer alignment.

**UAP\_POST\_INITIAL\_WAFER\_LOAD**

Perform AutoZ after the first wafer load.

**NOTE** *AutoZ is a function of the SofTouch optional licensed feature for KTE. For more information on SofTouch, refer to the Prober and Prober Drivers Manual.*

Loop through slots defined in the cassette plan.

Wafer Loop

**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 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" datapool pointer.

Load Wafer Description File.

Prepare a complete wafer plan for execution.

Load all the KTMs in wafer patterns.

Keep track via DataPool 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" datapool flag.

Write Wafer header to the LOT file.

**UAP\_WAFER\_BEGIN**

Execute a wafer plan.

Add result\_list, failed\_result\_list & 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 DataPool as "current\_wwp\_list".

LOOP: (Execute only tests that are enabled in wafer plan)

Reset "KI\_ktxe\_redo\_macro" datapool flag.

Reset "ktxe\_disable\_kdf" datapool 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 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 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" datapool flag is not set then

#### **UAP\_SUBSITE\_END**

#### **UAP\_SITE\_END**

    increment to next test.

Update "current\_wwp\_list" datapool 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 via "KI\_ktxe\_retest\_wafer" datapool flag.

    if so

        goto start\_wafer\_test

    else

        load next 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 next WAFER  
Align next WAFER

### UAP\_ALIGN\_ERROR

```

If cassette is complete...
    Clean up the 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 via tstdsl()
Set datapool 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 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 routine(s)

*Since the abort exit handler routine is the last to execute, if the user wishes to do prober or LPT calls, they will have to perform a `tstsel` call first and then a `tstdsl` call to release.*

*Also, keep in mind 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 UAP point.*

### **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.

## **User Library files required for KTXE execution**

During 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 UAP user library. Example “uuuu” equals “KTXEAddIn” for `libKTXEAddIn.so`, `libkittKTXEAddIn.so`, and `KTXEAddIn_proto.h`.

## **The KTXEErrorHandler function**

This function will output additional information to the user. A data pool item, `ktxe_error_gui`, can be set by the user, 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 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
```

to enter the events (-ev) and errors (-e) into the logfile file.log.

To accomplish the same event and error logging with the environment variables, enter:

```
prompt> setenv KI_KTXE_EVENT_LOG           /mydir/file.log
```

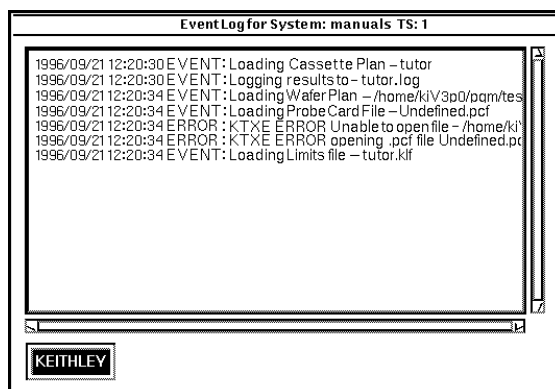
```
prompt> setenv KI_PRB_AUDIT_LOG           /mydir/file.log
```

```
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 [Figure 3-6](#).

**Figure 3-6**  
**Event log window**



## Lot suspend/resume

When the Abort button is pressed on the Status Dialog window, the operator can abort immediately, suspend, or cancel. If suspend is selected, 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 subsite 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 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.

## Multi-cassette Multi-lot Testing Utility (multi\_cassette)

The multi-cassette multi-lot 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 UAP module `KTXEExtSetWaferId` from the `KTXEAddIn` library to be placed at the `UAP_WRITE_LOT_INFO`.

Also, to disable the standard operator dialogs 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

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` datapool 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` datapool value. Or, a wafer id is generated in the form "Wafer\_nn", where "nn" is the cassette slot number.

At `UAP_VALIDATE_OCR` the datapool value `prober_wafer_id` can be checked or altered. See Appendix D for more datapool usage information.

## Abort flags

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 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, the user has access to the failed results list and the abort action flag.

At the user access point `UAP_HANDLE_ABORT`, the user can extract useful information regarding the failed results and determine the highest level of abort (i.e., 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, the user 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 the user wants to take some action based on the abort level, he can do it at the `UAP_HANDLE_ABORT`. For example, if the user wants to take different actions at different levels of abort, he can write the following code in a 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. The user can enable event logging by using the `-ev` flag at the command line when starting 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 for the user, he also has 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 KULT module that the user 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_usrplib.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
datapool */
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 (i.e., KTM). 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 the user wants to preserve the values of the abort flag or the failed results list, he must create his 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 LFE documentation 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 KTM by 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;
```

# 4

## Data Analysis

---

## Introduction

This section covers the tools required to analyze results data once testing has been completed. In this section the following will be discussed:

- **Data output formats** — This section will discuss the different formats that the results can be reviewed in.
- **Keithley Summary Utility (KSU)** — This section will cover how to use the features within KSU to analyze your results data.
- **Keithley Curve Analysis Tool (KCAT)** — This section will cover how to use KCAT to analyze your results data.

When you have completed this section you will be able to utilize KSU and KCAT to analyze your test results data and generate the desired report.

## Data output formats

There are two different ways to view test results from within the 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. This section will give an explanation of the output generated by these two software tools.

Also included is conversion utility to export data from the Keithley Data File 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)

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 software 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.
- Use the browser utility to search for lots based on a series of search criteria.

All these features are provided in an easy-to-use graphical user interface.

## Summary reports

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 [Figure 4-1](#). The report header displays information about the lot, process, and device. The header lines are displayed on each report page.

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 various 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 will reflect the abort data. In fact, the values that are out of range will be flagged to show that they meet the criteria required to abort, which are prescribed in the limits file.

Figure 4-1

### Sample standard report header

```
Lot          : 149616                Lot Summary Report KDF V1.0                Page: 1
Process     :                       Operator      : witzke
Device      :                       Starttime   : 15-Feb-97 16:14
Testname    : testengine             System      : 400ux
Limits File : LimFile3               Teststation: 1
? = outside valid limits             Wafer File:
* = outside selected limits
# = critical result is outside selected limits
Wafers: 1
Sites : 3
```

## Standard report

The standard report form shown in Figure 4-2 consists of the report header field, included in all but export report forms, followed by the lot data. 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 various 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 Figure 4-2.

Figure 4-2  
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

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

These are the 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 that parameter.

**Min and Max**

These are the minimum and maximum for the parameter's data points.

**SDEV**

This is the standard deviation from the mean of the parameter's data points.

**%SDEV**

This is the standard deviation expressed as a percentage plus or minus from the mean.

**CNT**

This is the number of data points used in calculations.

**%Vld**

This is the percentage of data points that are within the valid range specified in the limits file. This will be 100.0% if no limits file was specified.

**%Spec**

This is 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 particular data point for a test is found as follows:

1. Find the appropriate Wafer and Site in the rows along the left of the summary report.
2. Read across the row until the appropriate parameter/test is found. This value is the desired data point.
3. The relative information for the other data points in this parameter/test can be found by reading down the column until the Mean, MIN, and MAX are found.

Figure 4-3 is a raw report generated from the same .kdf file as the standard report shown in Figure 4-2.

Figure 4-3  
Raw report

Lot Summary Report KDF V1.0											Page: 1	
Lot	: xxx										Operator	: kthmgr
Process											Starttime	: 22-Nov-94 22:38
Device											System	:
Testname	: trial4										Teststation	: 1
Limits File	: xxx										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 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.

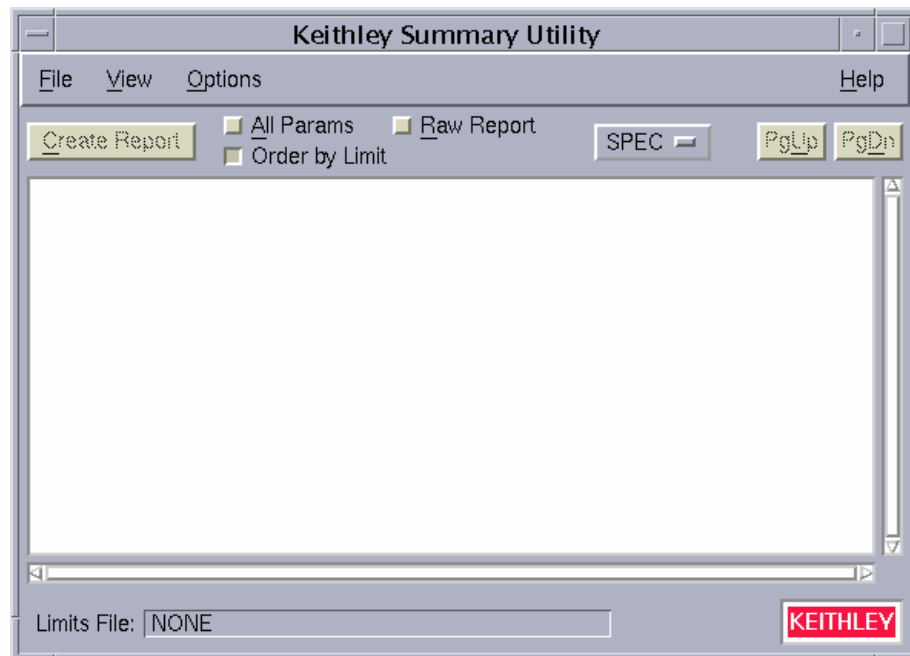
## Starting KSU

KSU can be started from the KSU icon or the command line. Some examples are:

- Double-click on the KSU icon or the executable file.
- Click on 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 [Figure 4-4](#) will be displayed.

*Figure 4-4*  
**KSU main window**



## KSU main window

The KSU main window menu bar contains four selections:

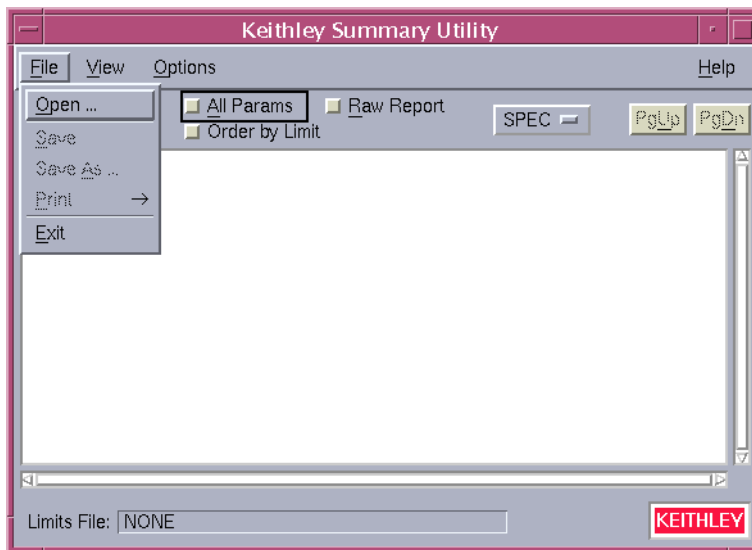
- File menu
- View menu
- Options menu
- Help menu

## File menu

Selecting File produces the menu shown in [Figure 4-5](#). The menu items are:

- **Open** — Produces the KSU Lot Browser window discussed below.
- **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 4-5  
KSU file menu



## KSU lot browser window

The Lot Browser window shown in [Figure 4-6](#) appears when you select Open from the File menu.

The lot browser allows data entry in all of the 14 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. [Table 4-1](#) 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.

The lot browser's main function is to search the .kdf file and produce reports based on that search. The user function controls can be found on the panel below the window that displays the available lot files.

Figure 4-6  
KSU lot browser window

Table 4-1  
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

### Lot browser controls

The lot browser functions are controlled with four screen buttons: Search, Clear, OK, and Cancel.

- **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 thereafter, you must press this button to initiate a search.
- **Clear** — Clears all search criteria and returns each field to the \* default.
- **OK** — Closes the 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 display in the Files that satisfy search criteria list box. To select a lot for search and report generation, click on the lot and click on the OK button, or double-click on the lot.

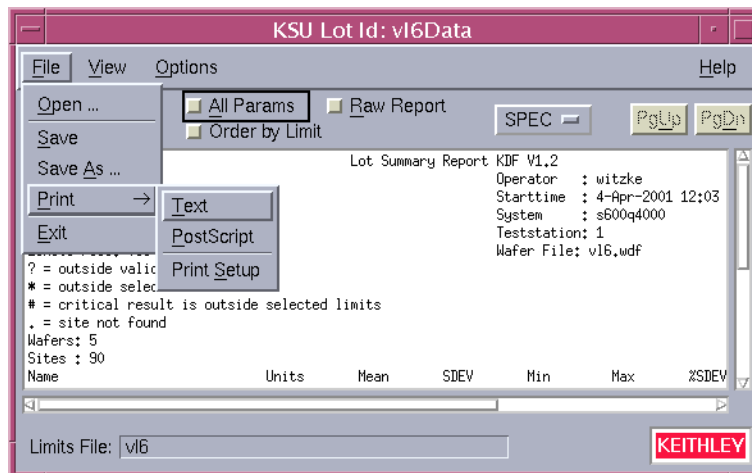
The Lot Browser window will close, and the selected lot will be transferred to the KSU window for processing. Selecting the Cancel button removes the Lot Browser window from the screen and returns control to the KSU main window.

## Print menu

When the Print option in the File menu is selected, the menu shown in [Figure 4-7](#) appears with the following options:

- **Text** — Prints to non-postscript printers according to formatting data in print setup.
- **PostScript** — Prints to postscript printers according to formatting data in print setup.
- **Print Setup** — Sets up page format for printing.

Figure 4-7  
KSU print menu



## Print Setup

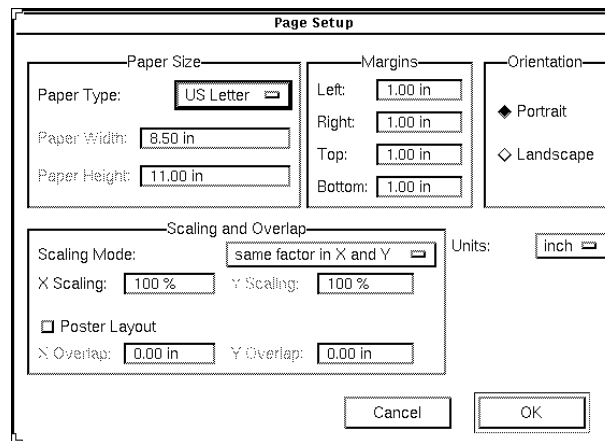
Use the Print Setup selection to set up the page format:

1. Select Print Setup from the Print menu. The Page Setup window shown in [Figure 4-8](#) appears.
2. Select the page options as desired:
  - 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)

Figure 4-8  
Page setup window



**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 will be changed by the same percentage.
  - Separate X and Y factors — The X and Y axes will be changed independently.
  - Fit to page — The X and Y axes will be changed accordingly to fit the paper size selected.
  - Fill one page (unequal) — The data will be sized to fill one page, and the X and Y axes will be unequal.
  - Fit page width — The data will be sized to fit the width of the paper size selected.
  - Fit page length — The data will be 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 choose Cancel to abort the printer setup operation.

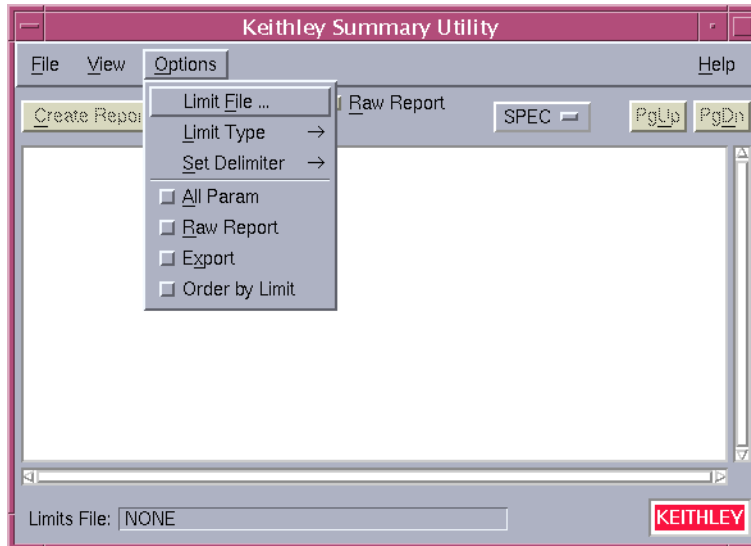
### View menu

Selecting View lets you select the KSU window's display format.

## Options menu

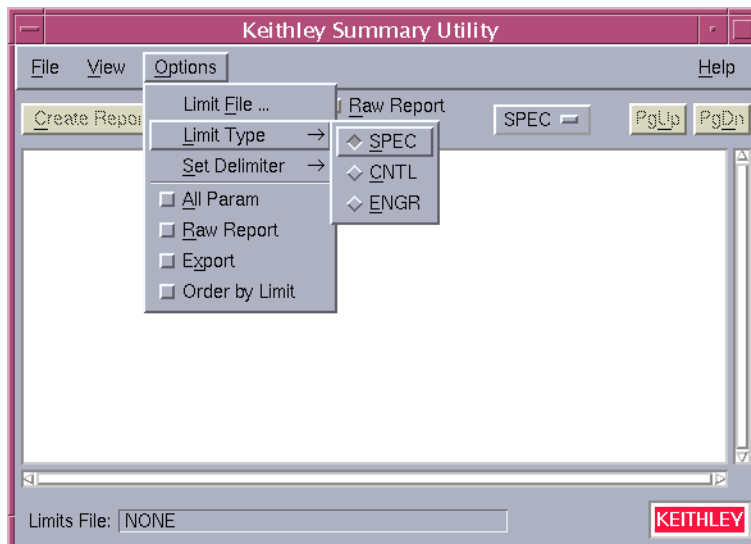
The Options menu shown in [Figure 4-9](#) contains the following selections:

Figure 4-9  
KSU options menu



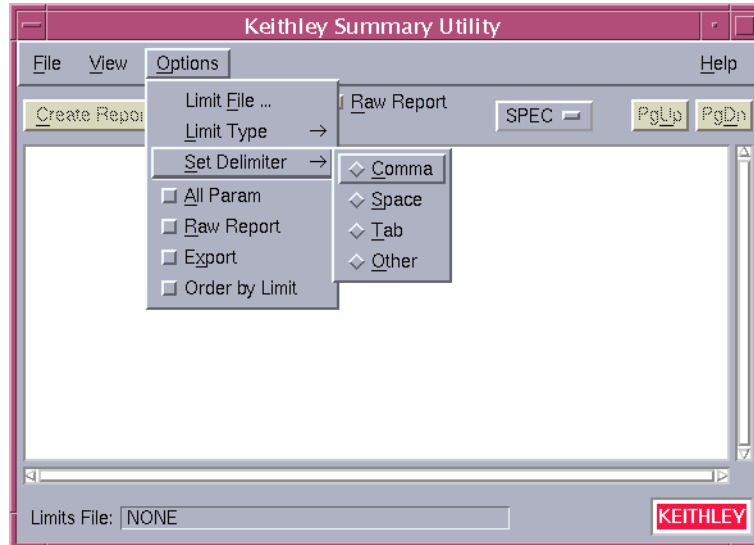
- **Limit File** — Lets you select a limit file to apply to the KSU.
- **Limit Type** — Lets you choose specific limits from the chosen limits file. Refer to [Figure 4-10](#).

Figure 4-10  
Limit type menu



- **Set Delimiter** — Lets you choose a delimiter to apply when the KSU file is exported using the Export option. Refer to [Figure 4-11](#).

Figure 4-11  
Set delimiter menu



- **All Param** — Includes all available parameters in the KSU report.
- **Raw Report** — Accesses all information contained in the selected lot file.
- **Export** — Lets you export the KSU report with the delimiter chosen using the Set Delimiter option.
- **Order by Limit** — Creates report results in the order specified in the limits file.

## Help menu

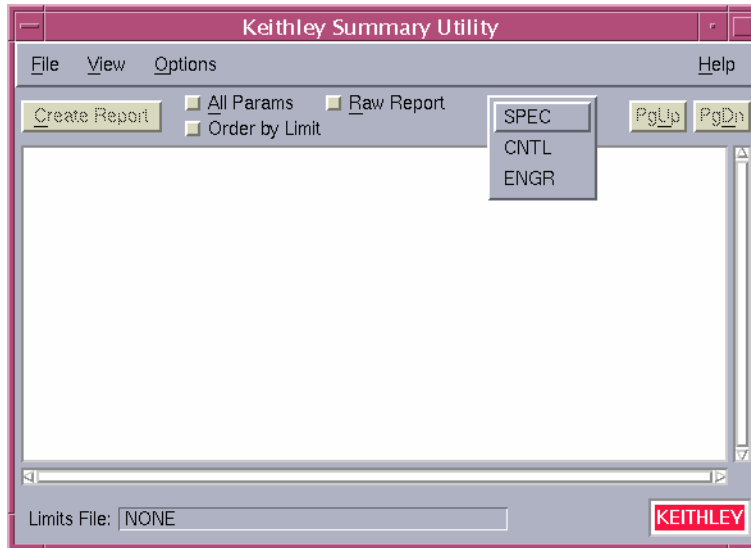
The Help menu item contains the following two selections:

- **KSU Documentation** — Provides help information on selected topics about KSU.
- **About** — Displays the installed KSU version number.

## Other KSU main window controls

The KSU main window contains the following six control selections. Refer to [Figure 4-12](#).

Figure 4-12  
Control selections



- **Create Report** — Initiates KSU report generation using the selected parameters.
- **All Params** — Includes all available data file information in the KSU report.
- **Order by Limit** — Directs KSU to produce its report in the same manner as the `-e` command line switch.
- **Raw Report** — Directs KSU to produce its report in the same manner as the `-r` command line switch.
- **SPEC, CNTL, ENGR** — Allows you to choose which parameter set to apply to KSU report generation.
- **PgUp and PgDn** — Lets you scroll through the displayed report one page at a time.
- **Scroll Bars** — The scroll bars found at the right side and bottom of the window that allow you to position the displayed file within the limits of a single displayed screen.
- **Limits File** — Displays the presently selected limits file.



## Command-line options for KSU

The following is a listing of the command-line options that can be entered when starting KSU.

- a Report on all parameters whether or not they are in the limits file.
- c Disable GUI interface (use 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 in which case it 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.

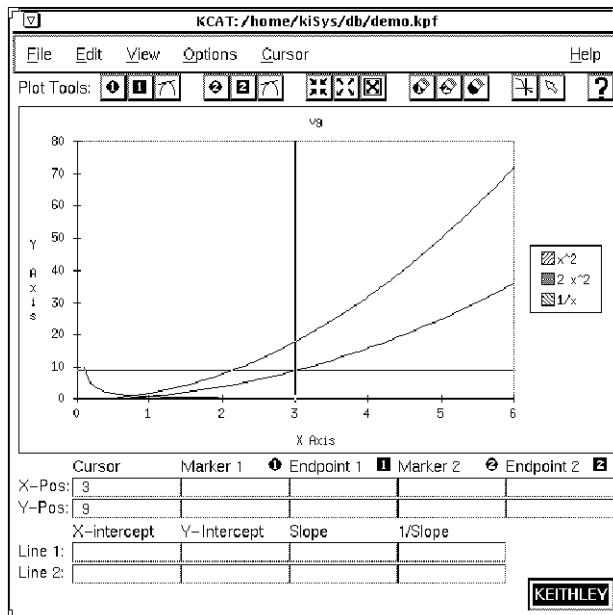
## Keithley Curve Analysis Tool (KCAT)

This section describes the windows and features associated with the Keithley Curve Analysis Tool (KCAT). For complete details on using the KCAT graphical interface, refer to the KCAT tutorial. The main function of KCAT is to graph results data and calculate slopes, intercepts, and tangents on the resulting curves.

### KCAT main window

Figure 4-13 shows the KCAT main window, which is the primary interface to the graphing capabilities of KCAT. When KCAT is executed, the main window appears in the upper left quadrant of the screen.

Figure 4-13  
KCAT main window



The main window has four sections:

- Menu bar
- Tool bar
- Graph area
- Data fields

## Menu bar

The menu bar contains six menus. The following list describes each of these menus.

**File** — Lets you load, save, or print Keithley plot files, and exit KCAT.

**Edit** — Allows customizing of the X and Y axis labels.

**View** — Allows you to display the full KCAT Main window, show the KCAT Data window, and/or the KCAT Scaling window.

**Options** — Lets you Select color or monochrome graphing, line or scatter graph, and the marker type.

**Cursor** — Lets you customize how the cursor interacts with the graphing area.

**Help** — Displays KCAT documentation and 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 and 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 [Figure 4-14](#), lets you manipulate the graph by turning on various check boxes or by scaling the range of the axes.

Check boxes associated with the Scaling window include:

- **Log check box** — Lets you change either the X axis data or the Y axis data to the logarithmic equivalent.
- **Invert check box** — Lets you invert either the X axis data or the Y axis data.

Figure 4-14  
Scaling window

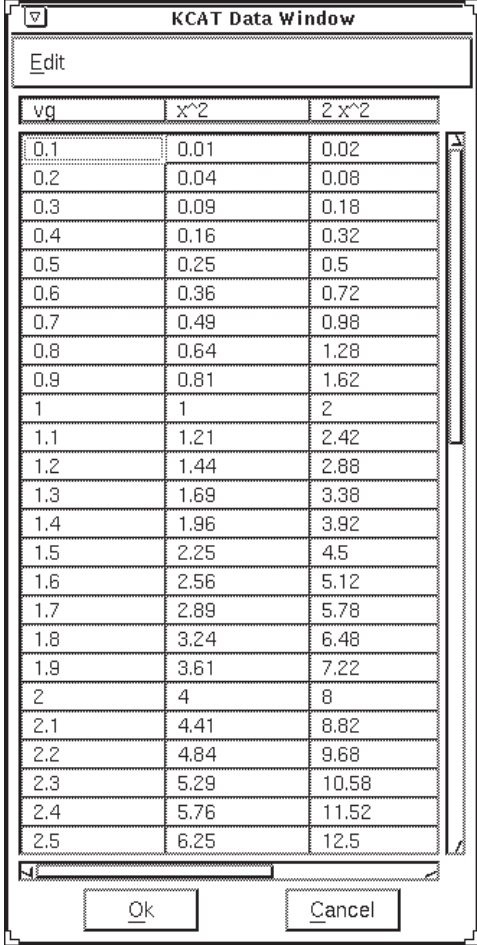
The image shows a dialog box titled "KCAT Scaling Window". It is divided into two main sections: "X Axis:" and "Y Axis:". Each section contains a "Log" checkbox and an "Invert" checkbox. Below these are "Min:" and "Max:" input fields. At the bottom of each section is an "Auto Scale" button. At the very bottom of the dialog are "Ok" and "Cancel" buttons.

Axis	Log	Invert	Min	Max	Auto Scale
X Axis	<input type="checkbox"/>	<input type="checkbox"/>			Auto Scale X
Y Axis	<input type="checkbox"/>	<input type="checkbox"/>			Auto Scale Y

## Data window

The Data window, shown in [Figure 4-15](#), contains a listing of all the data points associated with the present graph.

Figure 4-15  
Data window



vg	$x^2$	$2x^2$
0.1	0.01	0.02
0.2	0.04	0.08
0.3	0.09	0.18
0.4	0.16	0.32
0.5	0.25	0.5
0.6	0.36	0.72
0.7	0.49	0.98
0.8	0.64	1.28
0.9	0.81	1.62
1	1	2
1.1	1.21	2.42
1.2	1.44	2.88
1.3	1.69	3.38
1.4	1.96	3.92
1.5	2.25	4.5
1.6	2.56	5.12
1.7	2.89	5.78
1.8	3.24	6.48
1.9	3.61	7.22
2	4	8
2.1	4.41	8.82
2.2	4.84	9.68
2.3	5.29	10.58
2.4	5.76	11.52
2.5	6.25	12.5

## 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 for those who wish to use a spreadsheet or other Cornerstone-compatible document reader to interpret and edit their data.

Syntax of command:

```
KDFtoKCS [-c category][-f substitution_string][-h]
          [-l limit_file][-o outfile][-p] kdf
```

where:

category	is the limits category of interest
substitution_string	is an optional string to be substituted into the kcs file wherever the data value is invalid
limit_file	is the name of an appropriate Keithley Limits File (klf)
outfile	is the name of the intended output kcs file
kdf	is the name of the kdf to be converted

The program first accesses the Keithley Limits File (as specified in the kdf or by using the -l command line option) to write the header row to the .kcs file. The user has 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 can be found in folder specified by the environment variable \$KIDB, and has the name of the lot as defined in the kdf (or, alternately, the name supplied using the -o option).

The command line options are as follows:

- c only 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, as long as they are comma- or whitespace-delimited within the category field of their entry in the klf.)

- f the 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).

- h displays a help file on the screen.

- l sets the limits filename.

- o sets the output kcs filename.

- p uses the parameters as they appear in the klf. (Default is to get the parameters from the first site entry in the kdf.)

Example:

Given the following kdf and klf:

4 lotData1.kdf	detailLimit.klf
TYP,KDF V1.0	#Keithley Parameter Limits File
LOT,lotData1	Version,1.0
TST,ktxe	File,/TestArea/detailLimit.klf
SYS,Q22	Date,01/01/1999
TSN,1	Comment,KLF
OPR,operator1	<EOH>
LMT,detailLimit	ID,icur
STT,01-Jan-1999 13:26	NAM,current
<EOH>	UNT,mA
Wafer_01,,1,1	RPT,1
1,-4,1	CRT,0
icur,1.3124e-10	TAR,0.0000e+00
<EOS>	AF,0
Wafer_01,,1,1	AL,0
1,-4,2	VAL,1.3100e-10,1.3125e-10
icur,1.3126e-10	SPC,-2.0000e-10,2.0000e-10
<EOS>	CNT,-2.0000e-10,2.0000e-10
<EOW>	ENG,-2.0000e-10,2.0000e-10
	<EOL>

Upon executing

```
KDFtoKCS lotData1
```

The following Cornerstone file would be generated:

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:

	A	B	C	D	E
1	Wafer	Site	Row	Col	current mA
2	Wafer_01	1	-4	1	1.31E-10
3	Wafer_01	1	-4	2	1.31E-10
4					

**NOTE** The extension was changed from kcs to csv in order for Excel to recognize the format.

# 5

# System Administration

---



## Introduction

This section covers setting up and using the Keithley Test Environment (KTE) software for use with the KTE system. This section contains the following:

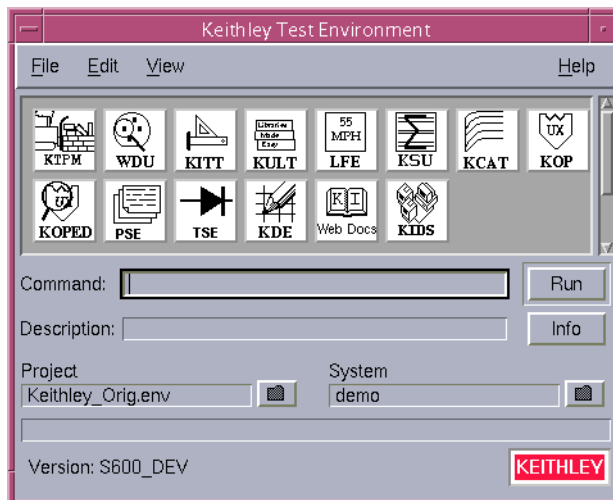
- **Workstation applications** — This will cover the Keithley Tool Palette, configuring the system software, file management, and customizing the workstation, for the KTE system.
- **Integrating the system into the customer network** — This section will cover integration of the KTE into the current system and logging test data.

## Workstation applications

### Keithley tool palette

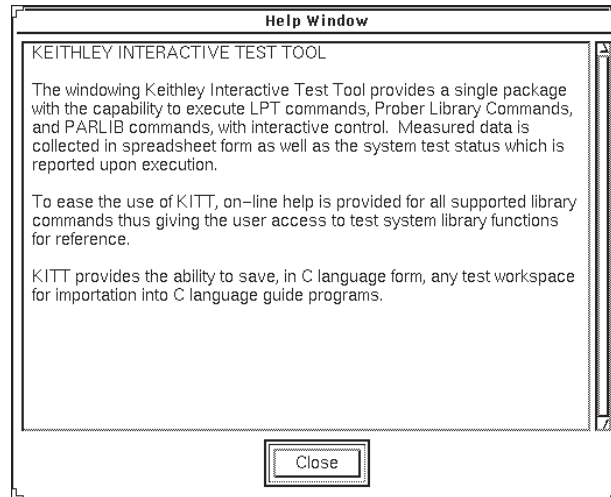
The tool palette provides a point and click interface into the KTE development environment. The tool palette for the S530 system, shown in [Figure 5-1](#), contain an icon for each of the test plan development and utility functions.

Figure 5-1  
Keithley tool palette



After clicking on an icon, the command related to that icon appears in the Command field and a brief description of the selected tool appears in the Description field. You can access more information about the selected command by clicking on the Info button located in the lower right corner of the tool palette. This will open the Help window shown in [Figure 5-2](#).

**Figure 5-2**  
**Tool window**



To start the currently selected tool, click on the Run button in the lower right corner of the tool palette or double-click on the icon of the software tool you want to initiate. The status of the program selected will appear 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** — Provides user modification access to the tool palette. The tool.tpi file controls the icons displayed by each of the View menu selections.
- **Reload tool.tpi** — Re-initializes 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, shown in [Figure 5-3](#), and lets you stop individual program execution.
- **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.

Figure 5-3  
Task list window

Program	User	pid	Start Time
kitt	kthman	237	10:07:42
kult	kthman	239	10:08:01

Buttons: Kill Task, Ok, Help

## Help menu

- **KTP documentation** — Lists version information about the KTE software.
- **About** — Opens the help window related to the software tool selected in the same manner as clicking on the Info button.

## Project field

The project field lets you select a specific project environment that describes the location of the KTE directory tree that will be utilized during testing.

## System field

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

## System configuration

### kth.ini file

The 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 will search three different locations for the kth.ini file:

- Your current working directory.
- Your 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 kth.ini 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 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 5 parameters per line, rather than its default setting of 10 per line. Perform the following:

1. Copy kth.ini 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 will ignore the setting and use a reasonable default value when it executes.*

4. Generate your Summary Report. The Summary Report Generator will use the new value of ParamsPerLine as long as you execute it from the working directory.

If you wish to make this change universal, replace the file in \$KIHOME with the new version.

## ktopm.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>
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,missmatch 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 desired
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
$KIHOM
$HOME

```

To modify the ktpm.ini file, use textedit or any other ASCII text editor program.

## Environment variables

The following list outlines the environment variables used by the various 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. Simply set the environment variables to the desired location(s) 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 in order for the change(s) 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

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

## LFE

Open:\$KI\_KTXE\_KLF  
 Read ID:Uses \$KI\_KTXE\_KTM

## WDU

Open: Uses \$KI\_KTXE\_WDF

## TSE

Open: \$KI\_KTXE\_TSF

## KCAT

Open: \$KI\_KTXE\_KDF

## PSE

Open: \$KI\_KTXE\_PSF

## KOPED

Open: Uses \$KIDAT

## KSU

Open: \$KI\_KTXE\_KDF

## KOP

.ini file loadUses \$KIDAT

The Keithley startup and login files define the following environment variables, shown in [Table 5-1](#) through [Table 5-7](#), for your use. 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.

*Table 5-1*  
**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

Table 5-2  
Open interface environment variables

Variable	Definition	Meaning
ND_HOME	\${KIHOME}openint	OI Home Directory
ND_PATH	\${ND_HOME}/lib:\$KIDAT	OI Search file path
ND_LIB	\${ND_HOME}/shr	OI Library subdirectory
OIT_LOOK	MOTIF	OI GUI "Look & Feel"
OIT_PATH	\${OIT_HOME}/lib:\$KIDAT	
OIT_LIB	\${OIT_HOME}/shr	
OIT_HOME	\${KIHOME}/openint	

Table 5-3  
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 KUI windows.	Provides a way to use existing KUI.
KI_ENBALE_2010_DISPLAY	Set to 1 to activate the front panel display of the optional 2010 DMM.	
KI_ENBALE_2410_DISPLAY	Set to 1 to activate the front panel display of the HV 2410 SMU.	
KI_DEFAULT_LIM_MODE	Set to "KI_INDICATOR" to use the KI_INDICATOR value (7.0e22) instead of the measured value.	
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 in order to have 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_SC_PRINT	Set to 1 to enable the SmartClear debug messages to be displayed.	



*Table 5-3*  
**System environment variables**

<b>Variable</b>	<b>Definition</b>	<b>Meaning</b>
KI_KELVIN_CHECK	Set to ON to enable or OFF to DISABLE kelvin check.	Undefined means disabled.

*Table 5-4*  
**Diagnostics environment variables**

<b>Variable</b>	<b>Definition</b>	<b>Meaning</b>
KI_PLATFORM	S530	Platform type
KI_DIAGTOOLS_CONFIG	\${KIHOME}/dat	Configuration file
KI_DIAGTOOLS_LOG	\${KIHOME}/log	Log output file

*Table 5-5*  
**User library tool environment variables**

<b>Variable</b>	<b>Definition</b>	<b>Meaning</b>
KI_KULT_PATH	\${KIHOME}/usrlib	Location of user libraries

**Table 5-6**  
**Test plan manager environment variables**

<b>Variable</b>	<b>Definition</b>	<b>Meaning</b>
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 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

**Table 5-7**  
**Log file environment variables**

<b>Variable</b>	<b>Definition</b>	<b>Meaning</b>
KI_KTXE_ERROR_LOG	Defined by user	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.

Table 5-7  
**Log file environment variables**

Variable	Definition	Meaning
KI_DISABLE_TRACE	Undefined	Will disable any IC log messages from being displayed. Note that if the KI_IC_AUDIT_LOG is defined, ERROR level messages will still be 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.	

## 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. See exithandlers.
- **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 will be placed in the file specified.

**NOTE** *If the "-e" and "-ev" command-line options for the execution engine have a filename specified, this filename overrides the environment variables.*

KI\_KTXE\_ERROR\_LOG — determines the path and filename of the 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 LPT 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 LPT 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 will log KTXE errors to /mydir/file.log.

## File management

The system disk on the 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 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** — KULT user libraries.

## KCM utility description

Use the 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)

ifilename of file to be imported. The file must be generated by kcm. Paths are supported.

efilename 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" will create a package that contains all KTE files necessary to execute the sample.cpf cassette plan.

Using "kcm -i sample.cpf -u" will install this package into the current environment.

## Project environments

The .ki\_setup file has been modified to dynamically adjust a user's environment variables on a project level basis. This allows users to set up different areas to store, run, and load their product specific files...or for the separation of production and development code.

Users may use the make\_project script to create a new tree containing, if desired, the following directories:

- db
- pgm
- plans
- usrlib

The script will prompt for each of these possibilities and create the directories as necessary. In the case of the usrlib directory, our original KIHOME/usrlib directory will be copied over to the new location.

The following example will create a new project named dev1Area and the tree "root" of the project will be 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, into the KIDAT directory.

The example above create 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, and then present an enumerated list, and ask which one you want. You may cancel by hitting 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 will set the following environment variables:

KIPROJ — Points to the base path of the project tree.

KI\_PROJ\_DB — Points to path of directory where KDF and KLF files are stored.

KI\_PROJ\_USRLIB — Points to usrlib directory where 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 WDF, and KTM files are stored.

KI\_PROJ\_NAME — Project Name.

The `.ki_setup` file will use these base variables to configure the remaining KTE environment variables.

A GUI window is provided in KOP and KTP. By changing the project within these tools, all applications started by these tools will use the project environment specified.

When you start a new shell, or log in, the last project environment saved in `$HOME/current.env` will be loaded. If this file does not exist, then the system will look for the existence of `$KIDAT/current.env`. If that does not exist, then the system will take 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 may wish to 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

### 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 number(s). You must also create a `$KIDAT/acconfig_QMO#.ini` file for each QMO listed in the `$KIDAT/kitester.dat` file.

The `select_tester` script will create a `.ki_define_config` file in the user's home directory. This file is then "source'd" if `select_tester` is executed interactively. If `select_tester` is called non-interactively, the user **MUST** source `~/ki_define_config` in order that the KTE environment is configured correctly.

#### `select_tester` script

The `select_tester` routine will enable 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 will be interactive. “select” is an alias for “source \${KIBIN}/select\_tester”

Non-interactive 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 S530 System Selection Utility:
```

```
-----
NumberQMOSystem 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>
```

```
-----
Selecting tester 99, the debug tester, will use the fake prober. The user will not control an actual tester or prober.
```

#### **.ki\_define\_config**

The select\_tester script creates a .ki\_define\_config file in the user’s home directory. This file is then “source’d”, if in interactive mode, to initialize the proper environment variables based upon the tester selected.

### S530 environment variables

The following environment variables are defined by select\_tester that allow the 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
```

\$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 via the use of a \$KIHOME/ki\_setup\_\$QMO file. This file, if it exists for a particular QMO number, is also “source’d” so that additional environment variables or other customizations can be initialized.

```
kitester.dat
```

The following sample kitester.dat file will allow 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

### Logging in the workstation

The 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 [Table 5-8](#).

*Table 5-8*  
**Login accounts**

Type of Account	Log-in name	Password	Login 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 has four major categories to be defined:

- Keithley-Tools
- Unix-Tools
- Unix Administration
- User Programs

These categories correspond to the four different 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 describe the contents for one entry within a category:

ICON#=iconfile,scriptfile,infofile,terminal,warning

- “#” is the number of the icon in the KTP. Icons are numbered sequentially, starting at 1.
- “iconfile” is 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” is 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” is 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 KITT. The valid entries for this input are Y for yes and N for no.
- “warning” is the flag that specifies whether the user 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 scriptfiles (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 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 (LPTLIB)**

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 (KITTT) and the Keithley User Library Tool (KULT). For more information about the commands, refer to the LPTLIB Programming manual.

## **Keithley Data Files (KDF) library**

KDF is a series of routines to organize and save parametric test data into simple ASCII data files. KDF is discussed further in Appendix A.

## **Keithley User Interface (KUI) library**

KUI contains a series of subroutines that support the creation of user interface dialogs for your test programs. Using KUI, you will be able to control and monitor test activities during the test process. For more information about the KUI subroutines, refer to Appendix B.

**A**  
**KDF**

---

## Test data logging

The following paragraphs describe control and handling of the test data.

### Keithley Data Files (KDF) library

The 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.

KDF does not make use of any special environment variables. It does, however, use 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 functions 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 KDF data file 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 [Table A-1](#).

*Table A-1*  
**KDF output description file**

<b>Data file record</b>	<b>Description of data</b>
TYP,KDFV1.0	Identifies the file as a valid V1.0KDF 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 [Table A-2](#).

*Table A-2*  
**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 (1-4)
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

## Data logging interfaces

Refer to the “Data logging routines” on page [A-12](#) for information on the different interfaces used to perform data logging.

## KDF 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 get a picture of which results met or exceeded the desired limits.

## 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. You may, however, 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 18 attributes that identify and process each measured test result. The following list describes each attribute:

1. ID, limit record ID tag
  2. NAM, limit record name
  3. UNT, limit record units
  4. CAT, category description
  5. RPT, report option: 1 = use in Summary Report, 0 = ignore
  6. CRT, critical level
  7. TAR, target value of parameter
  8. AF, abort flag setting
  9. AL, limit abort flag is to be compared to
  10. VAL, low and high VALID limits
  11. SPC, low and high SPECIFICATION limits
  12. CNT, low and high CONTROL limits
  13. ENG, low and high ENGINEERING limits
  14. ena, enabled flag (only for adaptive test): 1 = test, 0 = ignore
  15. cla, class data field
  16. usr1, user data field
  17. usr2, user data field
  18. 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 will contain a NULL pointer, not an empty string, if the data is not defined in the limits file. The NAM, UNT, and CAT fields will contain an empty string if the data is not defined in the limits file.

These eighteen attributes are repeated for each result ID in the limits file. The following limits file contains the file header and two separate result ID records:

```
Version,1.0
File,/home/kiSys/db/tutor.klf
Date,07/21/1996
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 [Figure A-1](#).

**Figure A-1**  
**Limits editor dialog window**

The screenshot shows the 'Limits Editor Dialog' window. The title bar reads 'Limits Editor Dialog'. The main area contains the following fields and controls:

- ID: moda\_n13x1\_vtati
- Category: test
- Critical Level: 1
- Name: moda\_n13x1\_vtati
- Units: volts
- Enabled for Measurement:
- Class: main
- Abort Action: None
- Report (Included in lot summary):
- Abort Limit: Valid
- Target: 1.32
- Valid: High: 1, Low: -1e+16
- Spec: High: 1e+16, Low: -1e+16
- Control: High: 1e+16, Low: -1e+16
- Engineering: High: 1e+16, Low: -1e+16
- Buttons: Prev Limit, Next Limit, Close
- User Field 1: user 1
- User Field 2: user 2
- User Field 3: user 3

## 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 = beta(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 p/rober 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 [Table A-3](#).

**Table A-3**  
**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	TYP,KDF V1.0	3,0,0	3,0,0
LOT,test1	LOT,test2	1,1.0000E+00	1,1.0000E+00
PRC,CMOS	PRC,CMOS	2,3.0000E+00	2,3.0000E+00
DEV,TNG-121	DEV,TNG-121	<EOS>	<EOS>
TST,Test1	TST,Test1	<EOW>	<EOW>
TSN,1	TSN,1	2,,0,0	2,,0,0
LMT,limits	LMT,limits	1,0,0	1,0,0
<EOH>	<EOH>	1,1.0000e+00	1,1.0000e+00
1,,0,0	1,,0,0	2,3.0000e+00	2,3.0000e+00
1,0,0	1,0,0	<EOS>	<EOS>
1,1.0000e+00	1,1.0000e+00	2,0,0	2,0,0
2,3.0000e+00	2,3.0000e+00	1,1.0000e+00	1,1.0000e+00
<EOS>	<EOS>	2,3.0000e+00	2,3.0000e+00
2,0,0	2,0,0	<EOS>	<EOS>
1,1.0000e+00	1,1.0000e+00	3,0,0	3,0,0
2,3.0000e+00	2,3.0000e+00	1,1.0000e+00	1,1.0000e+00
<EOS>	<EOS>	2,3.0000e+00	2,3.0000e+00
		<EOS>	<EOS>
		<EOW>	<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+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
	-	1.00e+15								

### Data retrieval using KDF

The following sample program demonstrates how to read data from a .kdf file. The program will attempt 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

```

## Data logging routines

### PutLot

Purpose	This routine will log the header information to the DB or FF. Lotadd is used to either append (Lotadd = APPENDLOT), create new (Lotadd = CREATELOT), or to replace an existing lot (Lotadd = CREATELOT). Opens a file, writes to it, and then closes it.
Format	Status = <b>PutLot</b> (*LotStruct, Lotadd)  LOT *LotStruct — the lot where data is logged.  int Lotadd — Can be set to either CREATELOT or APPENDLOT and determines what will happen if a lot file already exists.
Remarks	You cannot use the '*' or the '?' characters in the Lot structure fields because they are the wildcard characters. Use of these characters will result in an error.  The Lot ID will be used as the lot filename, with the '.kdf' extension added.  PutLot logs all the data up to and including the '<EOH>' marker.  If a lot file already exists, it will be renamed from a .kdf extension to a .kd% extension.
Example	Full code can be found in sample program 1.

```

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);

```

## PutWafer

Purpose	This routine will log the information in the wafer structure to the DB or FF for a specific instance of a lot. Opens a file, writes to it, and then closes it. Must be followed at some point by EndWafer.
Format	Status = <b>PutWafer</b> (*LotStruct, *WafStruct)  LOT *LotStruct — The lot where data is logged. WAFER *WafStruct — The wafer to log to the lot.
Remarks	After a call to PutWafer, you must make a call to EndWafer before calling PutWafer again.  You cannot use the '*' or the '?' characters in the Wafer structure fields because they are the wildcard characters. Use of these characters will result in an error.
Example	Full code can be found in sample program 1.  <pre> for (waferloop = 1;waferloop &lt;= 3;waferloop++) { sprintf(testwafer-&gt;id,"%i",waferloop); status = PutWafer(testlot,testwafer); if (status &lt; 0) return(status); </pre>

## PutSite

Purpose	This routine will log the information contained in the Site Structure to the DB or FF for a specific instance of a lot and wafer. Opens a file, writes to it, and then leaves it open for parameter data. Must be followed at some point by an EndSite.
Format	status = <b>PutSite</b> (*LotStruct, *WafStruct, *SiteStruct)  LOT *LotStruct — The lot where data is logged. WAFER *WafStruct — The wafer to log to the lot. SITE *SiteStruct — The site to log to the lot.
Remarks	After a call to PutSite, you must make a call to EndSite before calling PutSite again.



You cannot use the '\*' or the '?' characters in the Site structure fields because they are the wildcard characters. Use of these characters will result in an error.

**Example** Full code can be found in sample program 1.

```
for (siteloop = 1; siteloop <= 10; siteloop++)
{
sprintf(testsite->id, "%i", siteloop);
status = PutSite(testlot, testwafer, testsite);
if (status < 0)
return(status);
```

## PutParam

**Purpose** This routine will log the information contained in the Param Structure to the DB or FF for a specific instance of a lot, wafer, and site. Writes to the already open file.

**Format** Status = **PutParam**(\*LotStruct, \*WafStruct, \*SiteStruct, \*ParamStruct)

LOT \*LotStruct — The lot where data is logged.

WAFER \*WafStruct — The wafer to log to the lot.

SITE \*SiteStruct — The site to log to the lot.

PARAM \*ParamStruct — The parameter to log to the lot.

**Remarks** You cannot use the '\*' or the '?' characters in the Param structure fields because they are the wildcard characters. Use of these characters will result in an error.

EndSite should be called after the last parameter for the current site is logged.

**Remarks** Full code can be found in sample program 1.

```
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);
```

## PutParamList

**Purpose** This routine will log a list of param Structures to the DB or FF for a specific instance of a lot, wafer, and site. The next pointer set to NULL will signify the end of the list to be logged.

**Format** Status = **PutParamList**(\*LotStruct, \*WafStruct, \*SiteStruct, \*Param)

LOT \*LotStruct — The lot where data is logged.

WAFER \*WafStruct — The wafer to log to the lot.

SITE \*SiteStruct — The site to log to the lot.

PARAM \*ParamStruct — The linked list of parameters to log to the lot.

**Remarks** PutParamList makes a series of calls to PutParam as it traverses the linked list, so it has the same rules as PutParam.

After calling PutParamList, remember to free up the memory from the list (use a while loop with RemoveParam).

**Example** Full code can be found in sample program 2.

```
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);
```

## EndLot

**Purpose** This routine ends the logging of the current lot. It must be called before another lot can be logged.

**Format** status = **EndLot**()

**Remarks** EndLot must be called before PutLot can be called again, otherwise an error is generated.

The end of the lot is signified by the end of the lot file.

## EndWafer

**Purpose** This routine ends the logging of the current wafer. It must be called after a call to PutWafer.

**Format** status = **EndWafer**()

**Remarks** EndWafer must be called before PutWafer can be called again, otherwise an error is generated.

EndWafer write the '<EOW>' marker to the file.

## EndSite

- Purpose** This routine ends the logging of the current site. It must be called after a call to PutSite.
- Format** `status = EndSite()`
- Remarks** EndSite must be called before PutSite can be called again, otherwise an error is generated.
- EndSite write the '<EOS>' marker to the file.

## GetLot

- Purpose** This routine returns a NULL terminated list of lots, starting with LotStructGot, that match the criteria specified in LotStructWanted. The LotStructGot pointer should already point to a structure when the routine is called (i.e. LotStructGot = CreateNewLot). Wild Cards are supported in the wanted structure. Wildcards can not be entered in the integer fields. (A value of zero in an integer position is the same as '\*'.)
- Format** `Status = GetLot(*LotStructWanted, *LotStructGot)`
- LOT \*LotStructWanted — Lot structure containing the information on the lot to be retrieved. It can be very general (using wildcards) or very specific.
- LOT \*LotStructGot — Lot structure to which the found data is returned. It must be an allocated structure when it is sent to GetLot.
- Remarks** Note: The Lot ID is the only required field.
- Using wildcards may cause a noticeable decrease in performance in a directory with many files.
- If the status returned is greater than or equal to zero, then it is the number of Lots found. If it is less than zero, it is an error code.
- Example** Full code can be found in sample program 1.
- ```
gotlot = CreateNewLot();

strcpy(testlot->id, "test1");
/* Retrieve all the data that was just logged */
GetLot(testlot, gotlot);
```

## GetWafer

- Purpose** This routine returns a NULL terminated list of wafers, starting with WaferStructGot, that match the criteria specified in WafStructWanted for the specific (single) LotStruct. Wild Cards are supported in the wanted structure. The WafStructGot pointer should already point to a structure when the routine is called (i.e. WafStructGot = CreateNewWafer). Wildcards are not supported for the integer fields. (A value of zero in an integer position is the same as '\*'.) If an empty (NULL) wanted structure is passed in, the routine will return all wafers in the specified lot.

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Format  | <p>Status = <b>GetWafer</b>(*LotStruct, *WafStructWanted, *WaferStructGot)</p> <p>LOT *LotStruct — The specific lot in which the wafer should be found.</p> <p>WAFER *WaferStructWanted — Wafer structure containing the information on the wafer to be retrieved. It can be very general (using wildcards) or very specific.</p> <p>WAFER *WaferStructGot — Wafer structure to which the found data is returned. It must be an allocated structure when it is sent to GetWafer.</p> |
| Remarks | <p>If the status returned is greater than or equal to zero, then it is the number of Wafers found. If it is less than zero, it is an error code. The wanted structure must contain a valid string or wildcard for the “id” and “split” string items. A null string in either of these string items will not return a match.</p>                                                                                                                                                      |
| Example | <p>Full code can be found in sample program 1.</p> <pre>gotwafer = CreateNewWafer();  /* Get all the wafers in the lot */ strcpy(testwafer-&gt;id, ""); GetWafer(gotlot, testwafer, gotwafer);</pre>                                                                                                                                                                                                                                                                                 |

## GetSite

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | <p>This routine returns a NULL terminated list of sites, starting with *SiteStructGot, that match the criteria specified in SiteStructWanted for the specific (single) LotStruct and WafStruct. The SiteStructGot pointer should already point to a structure when the routine is called (i.e. SiteStructGot = CreateNewSite). Wild Cards are supported in the wanted structure. Wildcards are not supported for the integer fields. (A value of zero in an integer position is the same as ‘.’.)</p>                                                                    |
| Format  | <p>Status = <b>GetSite</b>(*LotStruct, *WafStruct, *SiteStructWanted, *SiteStructGot)</p> <p>LOT *LotStruct — The specific lot in which the wafer should be found.</p> <p>WAFER *WaferStruct — The specific wafer in which the site should be found.</p> <p>SITE *SiteStructWanted — Site structure containing the information on the site to be retrieved. It can be very general (using wildcards) or very specific.</p> <p>SITE *SiteStructGot — Site structure to which the found data is returned. It must be an allocated structure when it is sent to GetSite</p> |
| Remarks | <p>If the status returned is greater than or equal to zero, then it is the number of sites found. If it is less than zero, it is an error code.</p>                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Example | <p>Full code can be found in sample program 1.</p> <pre>gotsite = CreateNewSite();  /* Get all the sites in this wafer */ strcpy(testsite-&gt;id, ""); GetSite(gotlot, gotwafer, testsite, gotsite);</pre>                                                                                                                                                                                                                                                                                                                                                               |

## GetParam

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine 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. The ParamStructGot pointer should already point to a structure when the routine is called (i.e. ParamStructGot = CreateNewParam). Wild Cards are supported in the wanted structure. Wildcards are not supported for the integer fields. (A value of zero in an integer position is the same as '*'.)                                                                                                                                           |
| Format  | Status = <b>GetParam</b> (*LotStruct, *WafStruct, *SiteStruct, *ParamStructWanted, *ParamStructGot)<br><br>LOT *LotStruct — The specific lot in which the wafer should be found.<br>WAFER *WaferStruct — The specific wafer in which the site should be found.<br>SITE *SiteStruct — The specific site in which the param should be found.<br>PARAM *ParamStructWanted — Parameter structure containing information on the parameter to be retrieved. It can be very general (using wildcards) or very specific.<br>PARAM *ParamStructGot — Parameter structure to which the found data is returned. It must be an allocated structure when it is sent to GetParam. |
| Remarks | If the status returned is greater than or equal to zero, then it is the number of parameters found. If it is less than zero, it is an error code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| Example | Full code can be found in sample program 1.<br><br><pre>gotparam = CreateNewParam();  /* Get all the parameters in this site */ strcpy(testparam-&gt;id, ""); GetParam(gotlot, gotwafer, gotsite, testparam, gotparam);</pre>                                                                                                                                                                                                                                                                                                                                                                                                                                       |

## GetParamList

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine returns a NULL terminated list of parameters to *ParamStruct that are included in the *ParamStructList list. Wild Cards are supported at the parameter level.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| Format  | Status = <b>GetParamList</b> (*LotStruct, *WafStruct, *SiteStruct, *ParamStructList, *ParamStruct)<br><br>LOT *LotStruct — The specific lot in which the wafer should be found.<br>WAFER *WaferStruct — The specific wafer in which the site should be found.<br>SITE *SiteStruct — The specific site in which the param should be found.<br>PARAM *ParamStructList — List of parameter structures to be retrieved. Each parameter in the list can be very general (using wildcards) or very specific.<br>PARAM *ParamStruct — Head of the list of parameter structures to which the found data is returned. It must be an allocated structure when it is sent to GetParamList. |
| Remarks | If the status returned is greater than or equal to zero, then it is the number of parameters found. If it is less than zero, it is an error code.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |

## GetLotData

|         |                                                                                                                                                                                                                                                                                                                                                                                         |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine returns a tree structure of all the wafers, sites, and parameters in LotWanted. The returned list begins with the next field of the lot structure sent to the function.                                                                                                                                                                                                    |
| Format  | Status = <b>GetLotData</b> (*LotWanted)<br><br>LOT *LotWanted - put the specific lot data that you want to find in this structure and the found lot will be returned in the LotWanted->next field.                                                                                                                                                                                      |
| Remarks | The data is returned in LotWanted->next. Then the data follows a tree structure from there.<br><br>LotWanted->wafers points to the first wafer in the lot.<br><br>LotWanted->wafers->sites points to the first site in the first wafer.<br><br>LotWanted->wafers->sites->params points to the first parameter in the first site of the first wafer.<br><br>Wildcards are not supported. |
| Example | Full code can be found in sample program 1.<br><br><pre>gotlot = CreateNewLot();  strcpy(gotlot-&gt;id, "test1"); /* Retrieve all the data that was just logged */ GetLotData(gotlot);</pre>                                                                                                                                                                                            |

## MatchParam2Limit

|         |                                                                                                                                                                                                                                                                   |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine takes the list of parameters and matches them to the corresponding limit codes. Each parameter points to its corresponding limit code and each limit points back to the parameter. If no match is found for a parameter, the pointer is set to NULL. |
| Format  | Status = <b>MatchParam2Limit</b> (*ParamList, *LimitList)<br><br>PARAM *ParamList — List of parameters.<br><br>LIMIT *LimitList — List of limits to match to the above parameters.                                                                                |

## FileExist

|         |                                                                                                                                                                                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | Checks for the existence of a file in the current data directory (where the lot files are being stored). Returns TRUE (1) if the file is in the directory, FALSE (0) if the file is not in the directory. |
| Format  | Status = <b>FileExist</b> (filename).<br><br>char filename[] — Name of the file to find.                                                                                                                  |
| Remarks | The current data directory is determined by the value in the kth.ini file after "Datapath=".                                                                                                              |

## LotExist

|         |                                                                                                                                                                      |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | Checks for the existence of a lot file in the current data directory. Returns TRUE (1) if the lot is in the directory, FALSE (0) if the lot is not in the directory. |
| Format  | Status = <b>LotExist</b> (*LotStruct) .<br><br>LOT *LotStruct — The lot structure containing the information to be found.                                            |
| Remarks | The current data directory is determined by the value in the kth.ini file after “Datapath=”.                                                                         |

## GetStartTime

|         |                                                                                                                                           |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | Returns a time and date string in the format “DD-MM-YYYY hh:mm” where DD=day, MM=month, YYYY=year, hh=hour, and mm=minutes.               |
| Format  | <b>GetStartTime</b> (timestring)<br><br>char timestring[] — String to which the current time is returned. Must be at least 20 characters. |

## DeleteLot

|         |                                                                                                                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will delete all lot associated data for the specified lot structure. All Lots in the NULL terminated linked list will be deleted. The DELETEDLOT routine is responsible for ensuring referential integrity. |
| Format  | Status = <b>DeleteLot</b> (*LotStruct)<br><br>LOT *LotStruct — The lot or linked list of lots to be deleted.                                                                                                             |
| Remarks | Wildcard deletes are not allowed.<br><br>When a lot is deleted, the lot is renamed from the .kdf extension to a .kd% extension.                                                                                          |

## DeleteWafer

|         |                                                                                                                                                                                                        |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will delete all wafer information for the specified lot and wafer. All wafers for the NULL terminated linked list wafers will be deleted. Wild cards are not allowed in either structure. |
| Format  | Status = <b>DeleteWafer</b> (*LotStruct, *WafStruct)<br><br>LOT *LotStruct — The lot that contains the wafer to be deleted.<br>WAFER *WafStruct — The wafer or linked list of wafers to be deleted.    |
| Remarks | Wildcard deletes are not allowed.<br><br>When a wafer is deleted, the lot is renamed from the .kdf extension to a .kd% extension.                                                                      |

## DeleteSite

|         |                                                                                                                                                                                                                                                                                     |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will delete all site information for the specified lot, wafer, and site. All sites for the NULL terminated linked list of sites will be deleted. Wildcards are not allowed in any of the structures.                                                                   |
| Format  | Status = <b>DeleteSite</b> (*LotStruct, *WafStruct, *SiteStruct)<br><br>LOT *LotStruct — The lot that contains the wafer to be deleted.<br>WAFER *WafStruct — The wafer that contains the site to be deleted.<br>SITE *SiteStruct — The site or linked list of sites to be deleted. |
| Remarks | Wildcard deletes are not allowed.<br><br>When a site is deleted, the lot is renamed from the .kdf extension to a .kd% extension.                                                                                                                                                    |

## DeleteParam

|         |                                                                                                                                                                                                                                                                                                                                                                                          |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will delete the parameter information for the specified lot, wafer, site, and param. All params for the NULL terminated linked list of params will be deleted. Wildcards are not allowed in any of the structures.                                                                                                                                                          |
| Format  | Status = <b>DeleteParam</b> (*LotStruct, *WafStruct, *SiteStruct, *ParamStruct)<br><br>LOT *LotStruct — The lot that contains the wafer to be deleted.<br>WAFER *WafStruct — The wafer that contains the site to be deleted.<br>SITE *SiteStruct — The site that contains the parameter to be deleted.<br>PARAM *ParamStruct — The parameter or linked list of parameters to be deleted. |
| Remarks | Wildcard deletes are not allowed.<br><br>When a parameter is deleted, the lot is renamed from the .kdf extension to a .kd% extension.                                                                                                                                                                                                                                                    |

## DeleteLimitCode

|         |                                                                                                                                                                                                         |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine is used to delete entire sets of limits defined by a limit code. All limits specified in the NULL terminated list of Limitcodes will be deleted.                                           |
| Format  | Status = <b>DeleteLimitCode</b> (*LimitcodeStruct)<br><br>LIMITCODE *LimitcodeStruct — The limitcode or linked list of limitcodes to be deleted.                                                        |
| Remarks | Wildcard deletes are not allowed.<br><br>Limitcode information is used to generate the limits filename.<br>When a limitcode is deleted, the lot is renamed from the .klf extension to a .kl% extension. |



## DeleteLimit

|         |                                                                                                                                                                                                                                  |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine is used to delete limit records from the DB. All limits specified in the NULL terminated list of Limits will be deleted.                                                                                            |
| Format  | Status = <b>DeleteLimit</b> (*LimitcodeStruct, *LimitStruct)<br><br>LIMITCODE *LimitcodeStruct — The limitcode containing the limit to be deleted.<br><br>LIMIT *LimitStruct — The limit or linked list of limits to be deleted. |

## Update comment routines

### GetComment

|         |                                                                                                                                                                                  |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will fetch the comment from the .kdf file for a specific lot occurrence.                                                                                            |
| Format  | Status = <b>GetComment</b> (*LotStruct, comment)<br><br>LOT *LotStruct — The lot to retrieve the comment from.<br><br>char comment[] — The string where the comment is returned. |

### PutComment

|         |                                                                                                                                                                                                 |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will overwrite the comment in the .kdf file for a specific lot occurrence.                                                                                                         |
| Format  | Status = <b>PutComment</b> (*LotStruct, comment[])<br><br>LOT *LotStruct — The lot where the comment is to be changed.<br><br>char comment[] — The string sent to be logged as the new comment. |

## Update limits routines

### GetLimitCode

|         |                                                                                                                                                                                                                                                                                                |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine will fetch a list of limitcodes that match the criteria specified in the wanted structure.                                                                                                                                                                                        |
| Format  | Status = <b>GetLimitCode</b> (*LimitcodeStructwanted, *LimitcodeStructlist)<br><br>LIMITCODE *LimitcodeStructwanted — The limitcode information to search for in the logging directory.<br><br>LIMITCODE *LimitcodeStructlist — The returned list of limitcodes that were found in the search. |
| Remarks | Limitcode information is used to generate the limits filename.                                                                                                                                                                                                                                 |

Wildcards can be used in the \*LimitcodeStructwanted structure.

The return value is the number of limitcodes found or an errorvalue if it is less than zero.

## GetLimit

**Purpose** This routine will fetch a NULL terminated linked list of limit structures with the specified Limitcode and limit information. The head of the linked list of limits is returned in both the \*LimitStruct and in the LimitcodeStruct->limits fields.

**Format** Status = **GetLimit**(\*LimitcodeStruct, \*LimitStruct)

LIMITCODE \*LimitcodeStructwanted — The limitcode to retrieve the limits from.

LIMIT \*LimitStruct — The returned list of limits that were found in the limitcode.

**Remarks** Limitcode information is used to generate the limits filename.

The return value is the number of limits found or an errorvalue if it's less than zero.

## PutLimit

**Purpose** This routine will write a list of limits to the DB or FF for the Limitcode specified. If the Limitcode already exists, the new limits will overwrite and append.

**Format** Status = **PutLimit**(\*LimitcodeStruct, \*LimitStruct)

LIMITCODE \*LimitcodeStructwanted — The limitcode to log the limits to.

LIMIT \*LimitStruct — The list of limits to log to the limitcode.

**Remarks** If the Limitcode already exists, it will be renamed from a .klf extension to a .kl% extension.

## Structure handling routines

There is a version of each of the structure handling routines for every structure (LOT, WAFER, SITE, PARAM, and LIMITCODE).

### AddNewLimitCode

### AddNewLot

### AddNewWafer

### AddNewSite

### AddNewParam

Purpose This routine adds “new” to the list following “current.”

Format **AddNewLimitCode** (\*current, \*new)  
**AddNewLot** (\*current, \*new)  
**AddNewWafer** (\*current, \*new)  
**AddNewSite** (\*current, \*new)  
**AddNewParam** (\*current, \*new)

LIMITCODE \*current — Pointer to the current LimitCode.

LOT \*current — Pointer to the current Lot.

WAFER \*current — Pointer to the current Wafer.

SITE \*current — Pointer to the current Site.

PARAM \*current — Pointer to the current Param.

LIMITCODE \*new — The new LimitCode to be added.

LOT \*new — The new Lot to be added.

WAFER \*new — The new Wafer to be added.

SITE \*new — The new Site to be added.

PARAM \*new — The new Param to be added.

Example `LIMITCODE *current, *new;`  
`AddNewLimitCode(current, new);`

### CreateNewLmtCode

### CreateNewLot

### CreateNewWafer

### CreateNewSite

### CreateNewParam

Purpose This routine allocates the memory for and returns a pointer to the new LimitCode, Lot, Wafer, Site, or Param.

Format `LimitCodePtr = CreateNewLmtCode ()`  
`LotPtr = CreateNewLot ()`  
`WaferPtr = CreateNewWafer ()`  
`SitePtr = CreateNewSite ()`  
`ParamPtr = CreateNewParam ()`

LIMITCODE \*LimitCodePtr — Pointer to a LimitCode structure.  
 LOT \*LotPtr — Pointer to a Lot structure.  
 WAFER \*WaferPtr — Pointer to a Wafer structure.  
 SITE \*SitePtr — Pointer to a Site structure.  
 PARAM \*ParamPtr — Pointer to a Param structure.

Remarks This routine must be called before performing operations with LimitCodePtr, LotPtr, WaferPtr, SitePtr, or ParamPtr.

Example `LIMITCODE *LimitCodePtr;`  
`LimitCodePtr = CreateNewLmtCode();`

## FindFirstLmtCode

## FindFirstLot

## FindFirstWafer

## FindFirstSite

## FindFirstParam

Purpose This routine returns the first LimitCode, Lot, Wafer, Site, or Param that “current” points to in the list. NULL is returned if “current” is NULL.

Format `LimitCodePtr = FindFirstLmtCode (*current)`  
`LotPtr = FindFirstLot (*current)`  
`WaferPtr = FindFirstWafer (*current)`  
`SitePtr = FindFirstSite (*current)`  
`ParamPtr = FindFirstParam (*current)`

LIMITCODE \*LimitCodePtr — Pointer to a LimitCode structure.  
 LOT \*LotPtr — Pointer to a Lot structure.  
 WAFER \*WaferPtr — Pointer to a Wafer structure.  
 SITE \*SitePtr — Pointer to a Site structure.  
 PARAM \*ParamPtr — Pointer to a Param structure.

LIMITCODE \*current — Pointer to the current LimitCode.  
 LOT \*current — Pointer to the current Lot.  
 WAFER \*current — Pointer to the current Wafer.  
 SITE \*current — Pointer to the current Site.  
 PARAM \*current — Pointer to the current Param.

Example `LIMIT *LimitCodePtr, *current;`  
`LimitCodePtr = FindFirstLmtCode(current);`

**FindLastLimitCode****FindLastLot****FindLastWafer****FindLastSite****FindLastParam**

**Purpose** This routine returns the last LimitCode, Lot, Wafer, Site, or Param that “current” points to in the list. It returns NULL if “current” is NULL.

**Format** LimitCodePtr = **FindLastLimitCode**(\*current)  
 LotPtr = **FindLastLot**(\*current)  
 WaferPtr = **FindLastWafer**(\*current)  
 SitePtr = **FindLastSite**(\*current)  
 ParamPtr = **FindLastParam**(\*current)

LIMITCODE \*LimitCodePtr — Pointer to a LimitCode structure.

LOT \*LotPtr — Pointer to a Lot structure.

WAFER \*WaferPtr — Pointer to a Wafer structure.

SITE \*SitePtr — Pointer to a Site structure.

PARAM \*ParamPtr — Pointer to a Param structure.

LIMITCODE \*current — Pointer to the current LimitCode.

LOT \*current — Pointer to the current Lot.

WAFER \*current — Pointer to the current Wafer.

SITE \*current — Pointer to the current Site.

PARAM \*current — Pointer to the current Param.

**Example** LIMITCODE \*LimitCodePtr, \*current;  
 LimitCodePtr = FindLastLimitCode(current);

**FindNextLimitCode****FindNextLot****FindNextWafer****FindNextSite****FindNextParam**

**Purpose** This routine finds the next LimitCode, Lot, Wafer, Site, or Param after the position that “current” points to in the list. NULL is returned if “current” is at the end of the list.

**Format** LimitCodePtr = **FindNextLimitCode**(\*current)  
 LotPtr = **FindNextLot**(\*current)  
 WaferPtr = **FindNextWafer**(\*current)  
 Site = **FindNextSite**(\*current)  
 Param = **FindNextParam**(\*current)

LIMITCODE \*LimitCodePtr — Pointer to a LimitCode structure.

LOT \*LotPtr — Pointer to a Lot structure.

WAFER \*WaferPtr — Pointer to a Wafer structure.

SITE \*SitePtr — Pointer to a Site structure.

PARAM \*ParamPtr — Pointer to a Param structure.

LIMITCODE \*current — Pointer to the current LimitCode.  
 LOT \*current — Pointer to the current Lot.  
 WAFER \*current — Pointer to the current Wafer.  
 SITE \*current — Pointer to the current Site.  
 PARAM \*current — Pointer to the current Param.

Example     LIMITCODE \*LimitCodePtr, \*current;  
               LimitCodePtr = FindNextLimitCode(current);

## FindPrevLimitCode

## FindPrevLot

## FindPrevWafer

## FindPrevSite

## FindPrevParam

Purpose       This routine finds the previous LimitCode, Lot, Wafer, Site, or Param before the position “current” points to in the list. NULL is returned if “current” is at the beginning of the list.

Format       LimitCodePtr = **FindPrevLimitCode**(\*current)  
               LotPtr = **FindPrevLot**(\*current)  
               WaferPtr = **FindPrevWafer**(\*current)  
               Site = **FindPrevSite**(\*current)  
               Param = **FindPrevParam**(\*current)

LIMITCODE \*LimitCodePtr - pointer to a LimitCode structure.  
 LOT \*LotPtr — Pointer to a Lot structure.  
 WAFER \*WaferPtr — Pointer to a Wafer structure.  
 SITE \*SitePtr — Pointer to a Site structure.  
 PARAM \*ParamPtr — Pointer to a Param structure.

LIMITCODE \*current — Pointer to the current LimitCode.  
 LOT \*current — Pointer to the current Lot.  
 WAFER \*current — Pointer to the current Wafer.  
 SITE \*current — Pointer to the current Site.  
 PARAM \*current — Pointer to the current Param.

Example     LIMITCODE \*LimitCodePtr, \*current;  
               LimitCodePtr = FindPrevLimitCode(current);

## InsertNewLmtCode

## InsertNewLot

## InsertNewWafer

## InsertNewSite

## InsertNewParam

Purpose       This routine adds “new” into the LimitCode Lot, Wafer, Site, or Param list before “current.”

Format       **InsertNewLmtCode**(\*current, \*new)  
               **InsertNewLot**(\*current, \*new)

```

InsertNewWafer(*current, *new)
InsertNewSite(*current, *new)
InsertNewParam(*current, *new)

```

LIMITCODE \*current — Pointer to the current LimitCode.  
 LOT \*current — Pointer to the current Lot.  
 WAFER \*current — Pointer to the current Wafer.  
 SITE \*current — Pointer to the current Site.  
 PARAM \*current — Pointer to the current Param.

LIMITCODE \*new— The new LimitCode to be inserted.  
 LOT \*new — The new Lot to be inserted.  
 WAFER \*new — The new Wafer to be inserted.  
 SITE \*new — The new Site to be inserted.  
 PARAM \*new — The new Param to be inserted.

Example      LIMITCODE \*current, \*new;  
               InsertNewLmtCode(current,new);

## RemoveLimitCode

## RemoveLot

## RemoveWafer

## RemoveSite

## RemoveParam

Purpose        This routine removes the LimitCode, Lot, Wafer, Site, or Param pointed to by “current” and returns a pointer to the next LimitCode, Lot, Wafer, Site, or Param in the list. If the next pointer is NULL, the previous LimitCode, Lot, Wafer, Site, or Param is returned. If the previous LimitCode, Lot, Wafer, Site, or Param is also NULL, then NULL is returned.

Format        LimitCodePtr = **RemoveLimitCode**(\*current)  
               LotPtr = **RemoveLot**(\*current)  
               WaferPtr = **RemoveWafer**(\*current)  
               SitePtr = **RemoveSite**(\*current)  
               ParamPtr = **RemoveParam**(\*current)

LIMITCODE \*LimitCodePtr — Pointer to a LimitCode structure.  
 LOT \*LotPtr — Pointer to a Lot structure.  
 WAFER \*WaferPtr — Pointer to a Wafer structure.  
 SITE \*SitePtr — Pointer to a Site structure.  
 PARAM \*ParamPtr — Pointer to a Param structure.

LIMITCODE \*current — Pointer to the current LimitCode.  
 LOT \*current — Pointer to the current Lot.  
 WAFER \*current — Pointer to the current Wafer.  
 SITE \*current — Pointer to the current Site.  
 PARAM \*current — Pointer to the current Param.

Example      LIMITCODE \*LimitCodePtr, \*current;  
               LimitCodePtr = RemoveLimitCode(current);

## LimitExist

Note: This routine is only for the LIMITCODE structure.

|         |                                                                                                                                                                                                                     |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | This routine tests for the existence of a Limit File in the current Limit File data directory based on the LimitCode "limitin." TRUE (1) is returned if the Limit Files exist, and FALSE (0) is returned otherwise. |
| Format  | Status = <b>LimitExist</b> (*limitin)<br><br>int Status — the result value of the call.<br><br>LIMITCODE *limitin — The LimitCode to search for.                                                                    |
| Example | int Status;<br>LIMITCODE *limitin;<br>Status = LimitExist (limitin);                                                                                                                                                |

## KDF user tag data

The 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 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.

Four routines have been added to the KDF library: PutTag, GetTag, PrintTagList, and ClearTagList. The 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. Max length is PARAM\_ID\_LENGTH, or 128 characters. tagName must be null-terminated.



- valueString has a max 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" );
```

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 malloc'd and needs to be free'd 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 desired tag. If tagName is NULL, all tags will be returned in the list else 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 malloc'd by the GetTag routine and must be free'd 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 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,s530q4000
TSN,1
OPR,williamson
STT,2-Nov-1999 13:23
WDF,sample.wdf
<EOH>
<TAG>"LotType",CMOS
<TAG>"LotPlant",Cleveland
Wafer_01,,1,1
<TAG>"WaferPlanUsed",WPFbase
Site_1,1,1
Parm1, 5.0000e+00
<EOS>
<EOW>
```

## UAP points

The following is a list of UAP points within KTXE and when the KDF logging routines are executed. The PutTag routine can be called anytime 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

### 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. (Doesn't 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. (Doesn't 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);
}
}
```



# B

## Keithley User Interface

---

## Introduction

The Keithley User Interface Library contains functions 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` for the library is provided. The following line should be placed in every test program or file which will be linked with the kui library:

```
#include "kui_proto.h"
```

Starting with KTE v5.1.0 and higher, the following KUI window routines will be displayed on the "Dialog" Tab on the KIDS interface. If this behavior is not desired, set the `KI_KUI_CLASSIC` environment variable to 1 to cause KUI routines to use the old-style windows.

If the KIDS interface is not running or connections are not being accepted by KIDS, KUI will default to the "Classic" windowing mode.

Note that the `WfrldDlg` window is not supported by the KIDS interface. The "Classic" windows (KOP) will be used for `WfrldDlg` routines.

## User interface constants

**DLG\_ABORT**  
**DLG\_EXIT**  
**DLG\_NO**  
**DLG\_OK**  
**DLG\_SKIP**  
**DLG\_YES**

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\_LOOK\_MOTIF**  
**DLG\_LOOK\_MSW**  
**DLG\_LOOK\_OPENLOOK**  
**DLG\_LOOK\_PM**  
**DLG\_LOOK\_PM2**

Constants which can be used to determine the look and feel of the user interface dialogs. They are intended for use with the `InitUI` function.

**FIELD\_ENABLED**  
**FIELD\_DISABLED**

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 function call.

**KI\_ABORT**  
**KI\_CONTINUE**  
**KI\_PAUSE**

Constants intended for use with the pause, continue, and abort flag (`pca_flag`) used by the Status dialog. Refer to the `StatusDlg` function.

**NUM\_LOT\_FIELDS**

Constant which specifies the number of fields within the Lot Information Dialog (LotDlg). Used to declare the `lot_dlg_fds` array with the proper dimension.

**MAX\_DLG\_LOOK**

Constant that indicates the highest value allowed for dialog look selection. Refer to the `GetProgramArg` function `gui look` command line argument switch and the `InitUI` function.

**EXIT\_LOT\_DLG****OPERATOR****LOT\_ID****PROCESS****DEVICE****TEST\_NAME****SYSTEM\_ID****TEST\_STATION****SEARCH\_KEY1****SEARCH\_KEY2****SEARCH\_KEY3****LIMIT\_FILE****LOT\_COMMENT**

Enums corresponding to fields in the Lot Information Dialog (LotDlg). Intended for use as indexes into the `lot_dlg_fds` array in order to select `FIELD_ENABLED` or `FIELD_DISABLED`. Refer to LotDlg.

## User interface library variables

**lot\_dlg\_fields[]**

Array which 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 which 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 KTXE is running with the KUI disabled, using the `Get_pca_flag()` routine will allow your UAP code to detect that a tester fatal event has occurred and can force the engine to exit. The `KTXEUpdateStatusAbort` 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 functions

### GetProgramArgs — Get Program Command Line Arguments

**Purpose** GetProgramArgs allows values for specific test program global variables to be passed in via the command line.

**Format**

```
void GetProgramArgs(int argc, char *argv[],
    int *debug,
    int *err_report_mode,
    char **err_log_fname,
    int *gui_look,
    LOT **lot,
    char **sum_report_options,
    char **kwf_fname,
    char *user_arg )
```

**Remarks** GetProgramArgs will parse the command-line arguments to match switches assigned to the test program variables. When a match is found, GetProgramArgs will then try to parse in it's argument, converting and bounding it per the variable's type, as required.

If an error occurs in interpreting the command-line arguments, GetProgramArgs will print an error message explaining the reason as well as the allowable command-line arguments to stderr, and exit the test program.

The allowable command-line arguments can be displayed by typing -h at the command line following the test program's name.

Valid switches are:

```
-c      text lot information comment field
-d      text lot information device field
-e n [fname] error reporting mode
          n:0-3
          0 - None
          1 - Display Error Messages
          2 - Log Error Messages
          3 - Display and Log Error Messages
          [fname] - error log file path and name
-g n    gui look and feel operation
          n:0-4
          0 - Motif
          1 - OpenLook
          2 - Microsoft Windows
-h      display command line options
-i      id lot information lot id field
-k n text lot search key
          n:1-3 field
-l id   lot information limit id
-o text lot information operator field
-p text lot information process field
-r "options" lot summary report options
```

```
-s text    lot information system field
-t n      test station
           n:1-4
-w fname  wafer description filename
-u text    user argument
-x n      debug flag
           n:-32767 - 32768
```

fname - valid filename and path

options - options must be enclosed in quotes

text - command line arguments will be concatenated to the switch argument until the next switch (dash-letter) occurs

### Example

The following illustrates how test program global variables can be set with default values within the program and optionally overridden from the command line:

```
/* KI_Strncpy guarantees the member will be xxx_LENGTH & null terminated */
KI_Strncpy(lot->limitcode, "tutorial_limits", LIMITCODE_LENGTH);
KI_Strncpy(lot->id, "tutorial_lot", LOT_ID_LENGTH);
lot->teststation = 1;
GetStartTime(lot->starttime);

if(getenv("USER")!=NULL)
KI_Strncpy(lot->operator, getenv("USER"), LOT_OPERATOR_LENGTH);

if(getenv("HOST")!=NULL)
KI_Strncpy(lot->system, getenv("HOST"), LOT_SYSTEM_LENGTH);

/*>> set default Keithley wafer (description) file name */
kwf_fname = "sample.wdf";

/*>> default lot summary report options */
sum_report_options = "-s";

/*>> set/override program run time values and flags w/ command line args */
GetProgramArgs(argc, argv, &debug, &err_report_mode, &err_log_fname,
&gui_look, &lot, &sum_report_options, &kwf_fname, user_arg);
```

## InitUINew — Initialize User Interface Library

|         |                                                                                                                                                                                                                                                                                                                               |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | InitUI is used to perform initialization so that dialogs can be displayed and become operable. It must be called before any dialogs are called. In addition, it also allows specifying the look and feel of the displayed dialogs as provided in the DLG_LOOK_XXXX constants. It also starts the control thread of execution. |
| Format  | InitUINew(int look, void *Main ( ) );                                                                                                                                                                                                                                                                                         |
| Remarks | The KUI system is now threaded for better response to operator input. As a result, the control thread process name is specified in the InitUINew function. The look program variable is defined in kui_proto.h. It is used with the GetProgramArgs function in order to set the look via the test program command line.       |

## InputDialog — Input Message Dialog

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | InputDialog displays a modal dialog window containing the passed message string and provides the user with a text edit widget to enter information into the program.                                                                                                                                                                                                                                                                |
| Format  | <code>int InputMsgDlg( char *msgstr, char *inputstr)</code>                                                                                                                                                                                                                                                                                                                                                                         |
| Remarks | The program remains in the dialog until the user presses one of the buttons. The text edit widget text is then placed in the inputstr buffer. The dialog then returns either DLG_OK or DLG_EXIT per the "OK" or "CANCEL" buttons. The input string pointer cannot contain a null value. It should point to some allocated memory space to place the entered string. If you do not want to use the string, pass '0' as the argument. |

## LotDlg — Lot Information Dialog

|         |                                                                                                                                                                                                                                      |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | LotDlg displays a modal dialog window in order to collect lot information from the operator. It has as arguments a pointer to a lot structure, a field enable array, and a bound for the maximum test station, that can be selected. |
| Format  | <code>LotDlg(LOT *lot, char lot_dlg_fields[NUM_LOT_FIELDS], int max_teststation)</code>                                                                                                                                              |
| Remarks | Default entries for the lot dialog fields can be passed through this structure prior to calling LotDlg as follows:                                                                                                                   |

(KI\_Strncpy guarantees the member will be xxx\_LENGTH & null terminated)

```

LOT *lot;
KI_Strncpy(lot->limitcode, "tutorial_limits", LIMITCODE_LENGTH);
KI_Strncpy(lot->id, "tutorial_lot", LOT_ID_LENGTH);
lot->teststation = 1;

```

"lot\_dlg\_fields" is an array passed to the lot information dialog to indicate the fields which can be altered by the user. The "lot\_fields" enum corresponds to indexes within that array and should be used along with FIELD\_ENABLED and FIELD\_DISABLED constants to set array elements as desired prior to calling LotDlg as follows:

```

lot_dlg_fields[SYSTEM_ID] = FIELD_DISABLED;
lot_dlg_fields[TEST_NAME] = FIELD_DISABLED;
LotDlg(lot, lot_dlg_fields, max_teststation);

```

All fields are enabled by default.

LotDlg returns either DLG\_OK or DLG\_ABORT per the "OK" and "ABORT" buttons. If exited with "OK," the dialog fields are updated and returned in the lot structure. The return value can then be used to determine further program execution.

Note: InitUI must be called before to LotDlg.

## OkCancelAbortMsgDlg — Ok Cancel Abort Message Dialog

|         |                                                                                                                                                                                                                                                                                                                                                                                                 |
|---------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | OkCancelAbortMsgDlg displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either “OK” or “CANCEL” before the program can continue.                                                                                                                                                                                                |
| Format  | <code>int OkCancelAbortMsgDlg( char *msgstr )</code>                                                                                                                                                                                                                                                                                                                                            |
| Remarks | The message should be phrased to state the test program can be aborted by pressing “CANCEL”. If “OK” is pressed, the dialog returns DLG_OK. If “CANCEL” is pressed, the user is then prompted through another modal dialog to verify aborting the test program. The dialog will then return either DLG_ABORT if “OK” was pressed, or DLG_NO if “CANCEL” was pressed in the verification dialog. |

## OkCancelMsgDlg — Ok Cancel Message Dialog

|         |                                                                                                                                                                                             |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | OkCancelMsgDlg displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either “OK” or “CANCEL” before the program can continue. |
| Format  | <code>int OkCancelMsgDlg( char *msgstr )</code>                                                                                                                                             |
| Remarks | Returns either DLG_OK or DLG_EXIT.                                                                                                                                                          |

## OkMsgDlg — Ok Message Dialog

|         |                                                                                                                                                                    |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | OkMsgDlg displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing “OK” before the program can continue. |
| Format  | <code>void OkMsgDlg( char *msgstr )</code>                                                                                                                         |
| Remarks | OkMsgDlg does not return a value. It is used to pause the test program and require the operator to acknowledge the message before the program can continue.        |

## QuitUI — Quit User Interface

|         |                                                                                                                                                  |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | It will remove any remaining displayed dialogs as well as perform clean-up actions required for using the dialogs in the User Interface library. |
| Format  | <code>int QuitUI()</code>                                                                                                                        |
| Remarks | QuitUI should be called prior to program exit.                                                                                                   |

## ScrollMsgDlg — Scrollable Message Dialog

|         |                                                                                                                                                                                              |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | ScrollMsgDlg is used to set a label of a dialog window.                                                                                                                                      |
| Format  | <code>void ScrollMsgDlg( char *label )</code>                                                                                                                                                |
| Remarks | Messages accumulate and can be scrolled through in the visible area until it is cleared or QuitUI is called. ScrollMsgDlg is called with a label to display at the top of the dialog window. |

Refer to UpdateModelessDlgs and UpdateStatusDlg for issues regarding the responsiveness of the ScrollMsgDlg.

## ScrollMsgDlgClr — Scrollable Message Dialog Clear

|         |                                                                                                                                |
|---------|--------------------------------------------------------------------------------------------------------------------------------|
| Purpose | ScrollMsgDlgClr is used to clear all messages from the scrolling message dialog.                                               |
| Format  | <code>void ScrollMsgDlgClr()</code>                                                                                            |
| Remarks | The scrolling message dialog will remain displayed and the first message sent will be placed at the top of the scrolling area. |

## ScrollMsgDlgMsg — Scrollable Message Dialog Message

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | ScrollMsgDlgMsg is used to post a message to the scrolling message dialog.                                                                                                                                                                                                                                                                                                                                                                                                                                               |
| Format  | <code>void ScrollMsgDlgMsg( char *msgstr )</code>                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| Remarks | The passed string should make use of “\n” as needed. If ScrollMsgDlg was not called previous to ScrollMsgDlgMsg, it will be called from within it, and the dialog label set as “Test Program Messages.” The buffer size is limited by the maxScrollLines variable which defaults to ~500 lines. When this buffer size is exceeded, the oldest 2/3 of the buffer is thrown away and the buffer will continue to grow. The max value can be adjusted via the datapool. Refer to the Datapool documentation for an example. |

## StatusDlg — Status Dialog

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | StatusDlg provides a modeless dialog window which will remain displayed from when StatusDlg is first called until QuitUI releases the user interface on program exit. Besides providing display fields for key test program variables, it also provides a single text display line on which a program-specific status message can be displayed as well as a level of program execution control through the “PAUSE,” “CONTINUE” and “ABORT” buttons.                                                                                                                                                                                                                                                                                               |
| Format  | <pre>void StatusDlg( LOT      **lot,                 WAFER   **wafer,                 SITE     **site,                 SUBSITE  **subsite,                 int       *total_wafers,                 int       *wafers_tested,                 int       *total_sites,                 int       *sites_tested,                 kui_support_t **KUI_Support,                 kui_user_t  **KUI_User );</pre>                                                                                                                                                                                                                                                                                                                                       |
| Remarks | StatusDlg was designed in conjunction with guide test program data. In order to minimize the argument list required for UpdateStatusDlg, StatusDlg establishes pointers to the test program variables from which the dialog fields will obtain their display information for the remainder of the test program. The status dialog treats all these variables as “read-only.” The lot, wafer, site, and subsite pointers point to structures with members corresponding to status dialog display fields. Since the addresses passed for these structures also point to the “present” structure in their respective linked list, the status dialog automatically track and display the correct information. The status dialog is called as follows: |



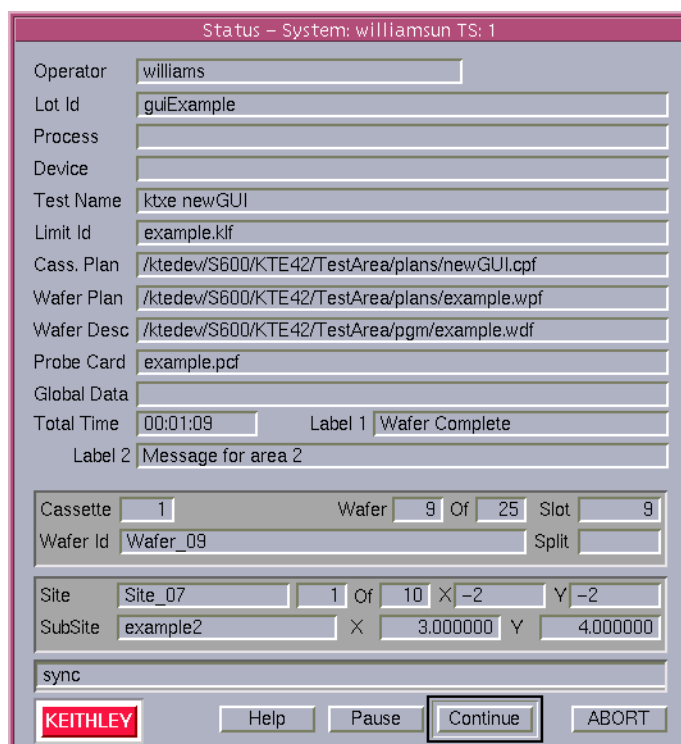
```
StatusDlg( &lot, &wafer, &site, &subsite,
           &total_wafers, &wafers_tested,
           &total_sites, &sites_tested,
           &KUI_Support, &KUI_User );
```

The status dialog contains a Total Time field. This field displays a running timer indicating elapsed time for test plan execution. Two user fields are also available for custom use.

There is a pointer to the KUI\_User structure in the datapool. Populating this structure will enable/disable the user fields. These fields must be initialized at UAP\_LOT\_INFO or earlier for proper operation. Once the fields are initialized, changing the values and calling the KTXEUpdateStatusAbort() function will cause an immediate update. If the user fields are left uninitialized, they will not be present on the Status dialog window.

Figure B-1 shows a sample status dialog window.

**Figure B-1**  
**Status dialog window**



## UpdateModelessDlgs — Update Modeless Dialogs

**Purpose** UpdateModelessDlgs is used to update and display the scroll message dialog when their contents change.

**Format** void UpdateModelessDlgs()

## UpdateStatusDlg — Update Status Dialog

**Purpose** UpdateStatusDlg is provided to update and display the status dialog fields when their contents change in the test program, as well as pass a status message appropriate for that point in the program.

**Format** `int UpdateStatusDlg( char *user_msg )`

**Remarks** Although the status dialog remains displayed throughout the test program, even while other dialogs are displayed, the fields are only updated and buttons checked when UpdateStatusDlg is called as in the following example:

```
UpdateStatusDlg("Loading Wafer ... ");
```

When UpdateStatusDlg is called, it first checks for differences between the test program variables and their display fields, updating them if necessary. The status message is then placed in it's field. The "PAUSE," "CONTINUE," and "ABORT" buttons are then checked.

The pause-continue-abort "pca\_flag" reflects the state of program execution. (By default, it is set to "KI\_CONTINUE".) When the status dialog buttons are checked by UpdateStatusDlg, if a button is pressed, it will set the flag appropriately.

Just before the UpdateStatusDlg function returns, it checks the pca\_flag. If paused, the test program will remain within the UpdateStatusDlg function call until either "CONTINUE" or "ABORT" are pressed. The pca\_flag will be set appropriately and allow the function to return with either DLG\_OK or DLG\_ABORT, respectively. The return value can be used to determine further program execution.

The responsiveness of updating the status dialog and the "PAUSE," "CONTINUE," and "ABORT" buttons is directly a result of where and how often UpdateStatusDlg appears in the test program. For example, if a number of tests which require significant time appear in the test program, it may be necessary to intersperse UpdateStatusDlg between them.

**NOTE** *InitUI must be called before StatusDlg. UpdateStatusDlg may appear in the test program before StatusDlg, but will be ignored as the status dialog pointers and the window itself have not been established.*

## VarMsgDlg — Variable Message Dialog

**Purpose** VarMsgDlg will create a window with a scrolling text region and multiple push buttons. In addition, each push button has a user-defined label.

**Format** `int VarMsgDlg( *VarMsgDlgDataPtr ) ;`

**Remarks** This routine is passed a structure containing configuration data for the window. The structure definition is shown below:

```
typedef struct _VarMsgDlgData
{
    int no_buttons;
    int no_lines;
    char **button_labels;
    char *win_label;
    char *ted_string;
```

```

}
VarMsgDlgDataRec, *VarMsgDlgDataPtr;

```

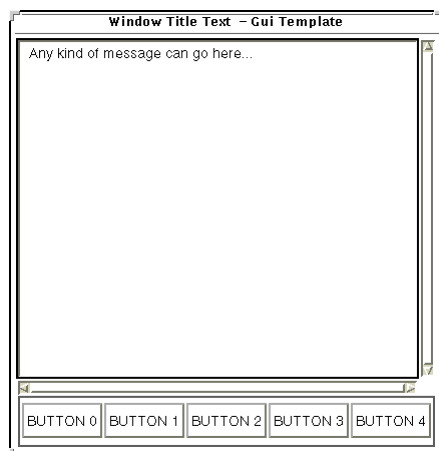
- **no\_buttons** — The number of pushbuttons displayed. There is no limit to the number of buttons, but screen space and size will impose a practical limit.
- **no\_lines** — How many lines of text will be displayed in the scrolling region.
- **button\_labels** — An array of text for labels.
- **win\_label** — The text used for the title bar on the window.
- **ted\_string** — The message to be displayed in the scrolling window.

This routine will return the index number of the button that was pressed by the user. The first button is index 0.

There is an example program located in the distribution showing sample KUI library calls and their uses. This file can be located in \$KIHOME/src/gui\_template.c.

Figure B-2 shows an example of a dialog window created using this routine.

**Figure B-2**  
**Variable message window**



## WfrIdsDlg — Multiple Wafer Information Dialog

**NOTE** *This window is not supported in the KIDS GUI. The standard KUI Classic window will be used.*

**Purpose** WfrIdsDlg displays a modal dialog window in order to collect information from the operator for multiple wafers to be tested by the test program. It facilitates automated testing as the wafer id will not have to be prompted as each wafer is tested.

It has as arguments a pointer to a wafer structure pointer, a bound for the highest cassette which can be selected, and a pointer to the variable which will contain the total number of wafers to be tested.

**Format** `int WfrIdsDlg(WAFER **wafer_ptr, int max_cassette, int *total_ptr)`

**Remarks** The multiple wafer information dialog allows the user to enter from 1 to max\_cassette lists of wafer ID's and split entries corresponding to 25 slots on each cassette. While the dialog is dis-

played, the user can switch between cassettes and can enter id's in any order. A split entry can only be made if the slot has a corresponding id entry.

The WfrIdsDlg is passed a pointer to the test program's "present" WAFER structure pointer. A pointer to the pointer is required to allow WfrIdsDlg to modify the test program's "present" WAFER structure pointer that is passed in and return a different pointer value.

For example, the wafer structure pointer may point to NULL when WfrIdsDlg is called, but points to an allocated wafer structure in the wafer linked list upon return.

As the user enters wafer information, the WfrIdsDlg will allocate elements and maintain a linked list of wafer structures. When the user exits the multiple wafer information dialog, the entered wafers will be in cassette-slot order in the linked list. The test program variable pointed to will contain the number of wafers in the linked list.

WfrIdsDlg returns either DLG\_OK or DLG\_ABORT per the "OK" and "ABORT" buttons. The return value can then be used to determine further program execution.

In most cases, the pointer to the wafer structure pointer will point to NULL when the dialog is called as the number of wafers, their position and ids will most likely be determined at program run-time. The default entries for the ID and Split fields will be empty. However, the ability to pass in a pointer to an established linked list of wafers is reserved for possible future development. It is intended for possible use to accommodate situations where wafer id's might remain the same from test run to test run, such as program development, where generic wafer id's are suitable, or for validation or editing of id's obtained through some other means such as optical character recognition or bar coding.

For this reason, the linked list of wafers passed to the dialog will reflect changes even if the dialog is exited with ABORT as no tracking of which wafers were passed in will be maintained by the dialog.

The multiple wafer information dialog is called as in the following example:

```
WfrIdsDlg(&wafer, max_cassette, &total_wafers);
```

**NOTE** *InitUI must be called before WfrIdsDlg.*

## WfrIdDlg — Single Wafer Information Dialog

**NOTE** *This window is not supported in the KIDS GUI. The standard KUI Classic window will be used.*

|         |                                                                                                                                                                                                                                                                                                               |
|---------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | WfrIdDlg displays a modal dialog window in order to collect information from the operator for single wafers to be tested by the test program. It is intended primarily for situations where an operator is prompted to enter information on a wafer by wafer basis, such as when manual wafer loading occurs. |
|         | It has as arguments a pointer to a wafer structure pointer, and a bound for the highest cassette which can be selected.                                                                                                                                                                                       |
| Format  | <code>int WfrIdDlg(WAFER **wafer_ptr, int max_cassette)</code>                                                                                                                                                                                                                                                |
| Remarks | The single wafer information dialog allows the user to enter the cassette number bounded from 1 to max_cassette, the slot from 1 to 25, the wafer id and split information. A split entry can only be made if the slot has an id entry.                                                                       |

A pointer to a WAFER structure pointer is passed to the WfrldDlg. A pointer to the pointer is used only to maintain similar arguments and reduce confusion with the multiple wafer information dialog WfrldsDlg which requires a pointer to a pointer. The passed pointer value will not be modified by WfrldDlg.

Unlike the multiple wafer information dialog, WfrldsDlg, the single wafer information dialog will not allocate a wafer structure if needed within the dialog call and expects to be passed a pointer to one.

Since it is passed a pointer to an allocated wafer structure, the dialog fields can be initialized based on the values of the members of the passed structure.

If the user exits the dialog with "OK," the structure members are updated with the dialog field entries and the WfrldDlg returns DLG\_OK. If "ABORT" is used to exit the dialog, the structure members are not updated and DLG\_ABORT is returned. The return value can then be used to determine further program execution.

The single wafer information dialog is called as in the following example, which allocates a wafer structure and initializes the fields based on the present entry in the wafer linked list, and adds it to the existing wafer linked list if the dialog is exited with "OK":

```
next_wafer = CreateNewWafer();
KI_Strncpy(next_wafer->split, wafer->split, WAFER_SPLIT_LENGTH);
next_wafer->boat = wafer->boat;
next_wafer->slot = wafer->slot;

if(next_wafer->slot < MAX_SLOT)
next_wafer->slot++;
switch( WfrldDlg( &next_wafer, max_cassette) )
{
case DLG_ABORT:
EXIT_PRGM
case DLG_OK:
AddNewWafer( wafer, next_wafer );
wafer = next_wafer;
wafers_tested++;
total_wafers++;
```

**NOTE** *InitUI must be called prior to WfrldDlg.*

## YesNoAbortMsgDlg — Yes No Abort Message Dialog

|         |                                                                                                                                                                                                                                                                                                                                                                                                                                                |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | YesNoAbortMsgDlg displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either "YES," "NO," or "ABORT" before the program can continue.                                                                                                                                                                                                                                           |
| Format  | <code>int YesNoAbortMsgDlg( char *msgstr )</code>                                                                                                                                                                                                                                                                                                                                                                                              |
| Remarks | The message should be phrased to state the test program can be aborted by pressing "ABORT." If "YES" is pressed, the dialog returns DLG_YES. If "NO" is pressed, the dialog returns DLG_NO. If "ABORT" is pressed, the user is then prompted through another modal dialog to verify aborting the test program. The dialog will then return either DLG_ABORT if "OK" was pressed, or DLG_NO if "CANCEL" was pressed in the verification dialog. |

## YesNoCancelMsgDlg — Yes No Cancel Message Dialog

|         |                                                                                                                                                                                                        |
|---------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | YesNoCancelMsgDlg displays a modal dialog window containing the passed message string and requires the user acknowledge it by pushing either “YES,” “NO,” or “CANCEL” before the program can continue. |
| Format  | <code>int YesNoCancelMsgDlg( char *msgstr )</code>                                                                                                                                                     |
| Remarks | Returns either DLG_YES, DLG_NO, or DLG_EXIT.                                                                                                                                                           |

## ContSkipAbortDlg — Continue Skip Abort Message Dialog

|         |                                                                                                 |
|---------|-------------------------------------------------------------------------------------------------|
| Purpose | ContSkipAbortDlg displays a message in a dialog box with the Continue, Skip, and Abort choices. |
| Format  | <code>int ContSkipAbortDlg ( char *msg )</code>                                                 |
| Remarks | Continue returns DLG_YES, Skip returns DLG_SKIP, and Abort returns DLG_ABORT.                   |

## LBoxDlg — List Box Message Dialog

**NOTE** *This window is not supported in the KIDS GUI. The standard KUI Classic window will be used.*

|         |                                                                                                                                                                                                           |
|---------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Purpose | The LBoxDlg is a window containing a list of items and allows selection of one or more of these items.                                                                                                    |
| Format  | <code>int LBoxDlg(char *text_label,<br/><br/>char *windowTitle,<br/>LBOXDLG_ListPtr **listPtr,<br/>int MultipleSelectionEnabled ) ;</code>                                                                |
| Remarks | <p><code>text_label</code> — text label over list of items<br/> <code>windowTitle</code> — title text for window frame<br/> <code>listPtr</code> — pointer to a linked list of LBOXDLG_ListPtr items.</p> |

```
typedef struct _lboxDlg
{
    char *label ;
    int selected ;
    struct _lboxDlg *next ;
} LBOXDLG_ListPtr ;
```

MultipleSelectionEnable – Single or Multiple Selection flag.

Possible values: LBOXDLG\_SINGLE\_SELECT or LBOXDLG\_MULTI\_SELECT

Return values:

```
DLG_OK      -Ok button pressed
DLG_EXIT    -Cancel button pressed
DLG_ABORT   -Abort button pressed
```

In addition, the listPtr list of items has been modified to reflect any selection changes. These changes only take effect if the “OK” button is pressed. Pressing the “CANCEL” or “ABORT” buttons leaves the list in its original state.

**Example** `int retVal, i ;`

```

LBOXDLG_ListPtr *lboxList, *new, *last ;

/* Create a list of items to display
*/
new = ( LBOXDLG_ListPtr *)malloc( sizeof( LBOXDLG_ListPtr ) ) ;
new->label = strdup( "Item 1" ) ;
new->selected = 1 ;
new->next = NULL ;

last = lboxList = new ;
for ( i = 2 ; i < 10 ; i++ )
{
    char label[ 64 ] ;

    sprintf( label, "Item %d", i ) ;

    new = ( LBOXDLG_ListPtr *)malloc( sizeof( LBOXDLG_ListPtr ) ) ;
    new->label = strdup( label ) ;
    new->selected = 0 ;
    new->next = NULL ;

    last->next = new ;
    last = new ;
}

/* Show list and get selection. ( Single select mode )
*/
retVal = LBoxDlg( "Test lbox text",
                 "Lbox Title",
                 &lboxList,
                 LBOXDLG_SINGLE_SELECT ) ;

/* free the list. Display the data while we are here
*/
while( lboxList != NULL )
{
    LBOXDLG_ListPtr *tmp ;

    if ( 1 == lboxList->selected )
        printf( "%s was selected!\n", lboxList->label ) ;

    free( lboxList->label ) ;
    tmp = lboxList->next ;

    free( lboxList ) ;
    lboxList = tmp ;
}

```

## KTE KUI localization

The `KI_LOCALIZE_CFG` environment variable is used to define an alternate localization file. Simply setenv `KI_LOCALIZE_CFG` to a file of the form below, and localization will automatically take place within KUI. Only the elements that you wish 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 gui windows. These values will have no effect in the 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 pathname 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"

KI\_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>
```

# C

## KTE File Formats

---

The following are examples of the data files that are created by the different tools in the KTE software. The extensions for each of the data files are:

- **.wdf** — The Wafer Description File created by WDU.
- **.tsf** — The Test Structure File created by TSE.
- **.ktm** — The Keithley Test Macro created by KITT.
- **.pcf** — The Probe Card File created by KITT.
- **.gdf** — The Global Data File created by KITT.
- **.psf** — The Parameter Set File created by PSE.
- **.klf** — The Keithley Limits File created by LFE.
- **.wpl** — The Wafer Plan File created by the Wafer Plan Editor in 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 tool (KRM).
- **.kdf** — The Keithley Data File created by an executed test, to be used by KSU.
- **.uap** — The UAP file used by KTPM and KTXE.
- **.kpf** — The Keithley Plot File created by KITT, to be used by KCAT.
- **.c** — The KULT module file. Used/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 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 (1-4). 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 KTXE; however, a 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 (1-8). 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-z, A-Z), digits (0-9), and the underscore character (“\_”).

This line is followed by any number of sites or projects to be associated with the pattern. Each line contains three fields:

1. The site name (for single-project wafers) or the project name (for multiple-project wafers) that is to be associated with the pattern.
2. The x coordinate.
3. 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 three fields:

1. The keyword "Site."
2. 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.
3. 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 three fields:

1. The subsite name. This can be any valid C-style string as described above.
2. The x coordinate of the subsite.
3. 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 filename (with .tsf extension) of the Test Structure File. The filename 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 three fields:

1. The data name. This string need *not* be a valid C-identifier; it may contain non-alphanumeric characters, including spaces. The only constraint is that it may not include commas.
2. 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.
3. 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 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 yy”. (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 three fields:
  1. The constant name. This string should be a local or global identifier specified under the KTM TEST MODULE VARIABLES banner.
  2. The datatype. This can be any type specified in kiox\_def.h, but it should correspond to the C-type the constant was declared as above.
  3. The constant type. This can be either “GLOBAL” or “LOCAL.”
  4. The value associated with the constant name.

- **KTM TEST MODULE PLOT AND LOG** — contains KITT configuration information for each of the test module variables (but *not* for the constants). Each line in this entry contains five fields:
  1. The variable name. This string should be an identifier specified under the KTM TEST MODULE VARIABLES banner.
  2. The variable plot code. This can be either “PLOT\_X” if KCAT should use the results of this variable as x values, “PLOT\_Y” if the user wishes KCAT to use the values of this variable as y values, or “PLOT\_OFF” if KCAT should ignore the variable.
  3. The variable log code. This can be either “LOG\_ON” if the user wishes the variable results to be written to a log file, or “LOG\_OFF” if the results should not be logged.
  4. A number.
  5. The variable user code. This can be either “USER\_ON” if the user wishes 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 three fields:

1. The pin name. This one-word string must start with a nonnumeric character, and must contain only letters (a-z, A-Z), digits (0-9), and the underscore character (“\_”).
2. 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).
3. 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 three fields:

1. The data name. This one-word string must start with a nonnumeric character, and must contain only letters (a-z, A-Z), digits (0-9), and the underscore character (“\_”).
2. The data type. All data types defined in `kiox_def.h` are supported.
3. 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\_parmset.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 non-numeric character, and must contain only letters (a-z, A-Z), digits (0-9), 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 three fields:

1. The parameter name. This follows the standard naming rules outlined above.
2. The data type. All data types defined in `kiox_def.h` are supported.
3. 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 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 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 non-alphanumeric 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 non-alphanumeric 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 KSU.
- **CRT** — contains 0 (zero) if the parameter is non-critical and any single digit number from 1-9 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 KTXE runs into an abort condition (see AL), 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 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 three 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-z, A-Z), digits (0-9), 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 two 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 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, UAMs could be used to distinguish these different modes of operation...switches are available to change what a single 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's 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 KDE (Keithley Data Editor) and read in by the KTXE. The data precedence is set to:
  1. PDI - local scope
  2. PCF data
  3. GDF data
  4. 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 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 three fields:

1. The slot ID. There are several options here. If the ID is absolute, then this field should simply contain the slot number. If the ID is relative, then 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.
2. 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-z, A-Z), digits (0-9), and the underscore character ("\_"). If the first field reads ALL or OPR, this field should be left blank.
3. The filename of the appropriate Wafer Plan File (with .wfp 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 KSU. Summaries could be per wafer plan (i.e. there could be different parameters, different limits, etc.). 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. Each line contains three fields:

1. The UAP name. This is the string that identifies the User Access Point. 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 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 the User to extend (customize) the Test Execution Engine (KTXE) by creating code to be inserted at designated spots (UAPs) by KTXE. The user 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 KTPM and can be found in the \$KI\_KTXE\_CPF directory.

The 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 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 can not 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>*

*<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 (1-4).
- **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 lists all the sites within a given wafer. One line is needed before getting into the site sections, and that line contains four fields:

1. 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-z, A-Z), digits (0-9), and the underscore character ("\_").
2. The wafer split. This optional one-word string must be a valid C-style identifier as described above.
3. The wafer boat. This optional one-word string must be a valid C-style identifier as described above.
4. 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 three fields:

1. The site ID. This one-word string must be a valid C-style identifier as described above.
2. The site row number.
3. The site column number.

This line is followed by a list of all the measurements which were taken for that site, one per line. Each line contains two fields:

1. The parameter ID. This is the C-style identifier string that matches a parameter ID in the Parameter Limits File.
2. 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, and can be found 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.

### UAPs

This section defines the User Access Points. Each line contains three fields:

1. The UAP name.
2. The user library name of the module the user wishes to use at this UAP.
3. 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 simply lists the variables specified in 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 PARMS: 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. (Incidentally, the KULT Module File name itself should be the name of the module with .c extension.)
- **MODULE RETURN TYPE** — contains the datatype of the module's return value. This is restricted to six types: char, float, double, int, long, and void.
- **NUMBER OF PARMS** — contains the number of parameters the module takes.

- **ARGUMENTS** — details information about each of the parameters, one per line. Each line contains six fields:
  1. 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-z, A-Z), digits (0-9), and the underscore character (“\_”).
  2. The datatype 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.
  3. Either “Input” or “Output,” depending on the purpose of the parameter.
  4. The default value of the parameter. This field is optional, but if it is used then the type should match the datatype in the second field. If the type in the second field is an array, then this field should be left empty.
  5. The minimum value of the parameter. This field is optional, but if it is used then the type should match the datatype in the second field. If the type in the second field is an array, then this field should be left empty.
  6. The maximum value of the parameter. This field is optional, but if it is used then the type should match the datatype 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 which 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 document, but not every C document is a KULT Module File. To be a KULT Module File, the C document must conform to the format explained above.

Test Macro Files are generated by KULT and can be found in the \$KI\_KULT\_PATH/*usr-lib* directory, where *usr-lib* is the name of the module’s User Library.

# D

## Data Pool

---

## Data pool

The data pool is used to hold global data while 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 the user to access global variables in KITT-generated test macros and also pass data pool items as parameters to KULT-generated modules that are run at UAPs.

For example, you could be using global data in your 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 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 (i.e., 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 [Table D-1](#) into the data pool:

*Table D-1*  
**Data pool items**

Name	Type	Initial Value	First Available	Last Usable
<b>KI_ktxe_redo_macro</b>	<b>INT</b>	<b>0</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
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.				
<b>KI_ktxe_retest_wafer</b>	<b>INT</b>	<b>0</b>	<b>UAP_WAFER_BEGIN</b>	<b>UAP_WAFER_END</b>
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.				
<b>KI_ktxe_skip_limits_check</b>	<b>INT</b>	<b>0</b>	<b>UAP_TEST_DATA_LOG</b>	<b>UAP_TEST_DATA_LOG</b>
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 <b>MUST BE SET</b> at or before UAP_TEST_DATA_LOG, else the default limit check routine could/would reset the abort flag(s).				

Table D-1  
Data pool items (cont.)

KUI_User	LONG_P	&KUI_User	UAP_PROG_ARGS	UAP_LOT_INFO
<p>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.</p> <p><b>Example:</b></p> <pre>#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 ; /* &lt;---- VERY IMPORTANT to de-reference tmp */  /* Initialize field 1 */ strcpy( user-&gt;label_1, "First Label " ) ; strcpy( user-&gt;data_1, "Data for first field" ) ;  /* Initialize field 2 */ strcpy( user-&gt;label_2, "Second Label " ) ; strcpy( user-&gt;data_2, "Data for second field" ) ;  KTXEUpdateStatusAbort( "Forcing update of display..." ) ;</pre>				
ManualProberType	INT	0	UAP_PROG_ARGS	UAP_PROBER_INIT
<p>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.</p>				
ManualWaferLoad	INT	0	UAP_PROG_ARGS	UAP_PROBER_INIT
<p>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.</p> <p><b>Example:</b> 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.</p>				
UAP_abort_level	INT	uap_abort_level	after a UAP is executed	after a UAP is executed
<p>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.</p> <p><b>Example</b></p> <p>A UAP description within a cassette plan file as:</p> <pre>UAP_TEST_END,myLib,UAP_abort_level=MyUAProutine()</pre> <p>will cause KTXE to exit if MyUAProutine returns KI_ABORT.</p>				

Table D-1  
Data pool items (cont.)

<b>abort_code</b>	<b>INT</b>	<b>-1</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
<b>abort_code_description</b>	<b>CHAR_P</b>	<b>&amp;abort_code_description</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
These two values are used by the UAP_ABORT_EXIT_HDLR routines. They can be set to user-defined values for abort handling.				
<b>abort_level</b>	<b>INT_P</b>	<b>&amp;abort_level;</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_ENGINE_EXIT</b>
Value that determines the abort level of a limit. <b>Example:</b> <pre> 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 ) ; </pre>				
<b>cl_kwf_fname</b>	<b>CHAR_P</b>	<b>cl_kwf_fname</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
Wafer description filename entered from "-w" command-line switch.				
<b>confirm_oper_wafers</b>	<b>INT</b>	<b>0</b>		
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.				



Table D-1  
Data pool items (cont.)

cpf_info	LONG_P	cpf_info	UAP_PROG_ARGS	UAP_ENGINE_EXIT
<p>Pointer to the cassette plan info structure.</p> <pre> typedef struct {     char version[MAXVERSIZE];     char cpfname[MAXFILENAME_SIZE];     char ascdatetime[MAXDATESTR_SIZE];     char comment[MAXCOMMENTSTR_SIZE];     char uapdefaults[MAXENGNAMESTR_SIZE];     char engine[MAXENGNAMESTR_SIZE];     char pcffname[MAXFILENAME_SIZE];     gdffname_list_t *gdffname_list;     char kdffname[MAXFILENAME_SIZE];     char wdffname[MAXFILENAME_SIZE];     slot_list_t *slot_list;     uap_list_t *uap_list; } cpf_info_t; </pre>				
current_slot_list	LONG_P	& current_slot_list	UAP_LOT_INFO	UAP_ENGINE_EXIT
<p>Pointer to the current_slot_list element. Updated each wafer.</p> <p><b>Example:</b></p> <pre> 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 ; </pre> <p><b>Structure definition:</b></p> <pre> typedef struct _slot_list {     char        slot[SLOTNAMESIZE];     char        wafer_id[MAXWAFERID];     char        split_id[MAXWAFERID];     char        wpfname[MAXFILENAME_SIZE];     char        plan_wpfname[MAXFILENAME_SIZE];     struct _slot_list *next; } slot_list_t ; </pre>				
current_wwp_list	LONG_P	current_wwp_list	UAP_SITE_CHANGE	UAP_SITE_END
<p>Pointer to the current node of the wwp list. Updated each Macro.</p> <p>Structure definition: See type definition for wwp_list_t in \$KIHOME/include/ktxe_types.h.</p>				

Table D-1  
Data pool items (cont.)

<b>cur_wwp_list_ptr</b>	<b>LONG_P</b>	<b>&amp;current_wwp_list</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
<p>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.</p> <p><b>Example:</b></p> <pre> 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-&gt;next != NULL ) {     while ( wwp-&gt;next != NULL )     {         if ( strcmp( wwp-&gt;next-&gt;siteid, "Site_3" ) == 0 )             break ;         else             wwp = wwp-&gt;next ;     } } /* adjust the current_wwp_list value */ *foo = (long) wwp ; </pre>				
<b>display_lotdlg</b>	<b>INT_P</b>	<b>&amp;display_lotdlg</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_LOT_INFO</b>
<p>Flag to enable/disable Lot Dialog display. Defaults to 1, or enabled. MUST be updated at or before UAP_LOT_INFO.</p> <p><b>Example:</b></p> <pre> int *logDlg ; /* disable the display of the Lot Dialog screen */ lotDlg = ( int *)dpGetPointer( "display_lotdlg", INT_P ) ; *lotDlg = 0 ; </pre>				
<b>failed_result_list</b>	<b>LONG_P</b>	<b>failed_result_list</b>	<b>UAP_HANDLE_ABORT</b>	<b>UAP_HANDLE_ABORT</b>
<p>Pointer to a results list containing results that failed the limits checking. Use this for any custom abort handling requirements.</p>				
<b>ktm_list</b>	<b>LONG_P</b>	<b>ktm_list_head</b>	<b>UAP_VALIDATE_OCR</b>	<b>UAP_WAFER_END</b>
<p>Pointer to the list of all KTMs used in a wafer plan. This is updated each wafer before UAP_WAFER_BEGIN.</p> <p><b>Structure definition:</b></p> <pre> typedef struct _ktm_list {     char ktmfname[MAXFILENAME_SIZE];     char wafpatname[MAXWAFPATNAME_SIZE];     struct _ktm_list *next; } ktm_list_t; </pre>				
<b>ktxe_abort_logging</b>	<b>INT_P</b>	<b>&amp;ktxe_abort_logging</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
<p>Flag to enable logging of the limits abort.</p>				
<b>ktxe_cpf_mode</b>	<b>INT</b>	<b>cassette plan probe mode</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
<p>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.</p> <p>ALL_MODE = 1 OPR_MODE = 2 ABS_MODE = 3 REL_MODE = 4</p>				

Table D-1  
Data pool items (cont.)

<b>ktxe_debug</b>	<b>INT_P</b>	<b>&amp;ktxe_debug</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
Debug flags from "-x" command-line switch.				
<b>ktxe_disable_exec_log</b>	<b>INT</b>	<b>undefined</b>		
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. Refer to the Recipe Manager documentation for additional information about the execution log.				
<b>ktxe_disable_kdf</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_END</b>
Flag for disabling standard kdf data logging.				
<b>ktxe_disable_ktm</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_LOT_END</b>
Flag for disabling macro test execution.				
<b>ktxe_disable_kui</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_LOT_END</b>
Flag for disabling user interface for KTXE.				
<b>ktxe_disable_load_cassette_msg</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROBER_INIT</b>
Flag for disabling "Load Cassette" message dialog.				
<b>ktxe_disable_plan_complete_msg</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
Flag for disabling the "KTXE Plan Complete" message.				
<b>ktxe_disable_prober</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_END</b>
Flag for disabling default KTXE prober routines.				
<b>ktxe_disable_statdlg</b>	<b>INT</b>	<b>0</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_LOT_INFO</b>
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.				
<b>ktxe_disable_uap</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
Flag for disabling execution of UAP routines.				
<b>ktxe_enable_cl_log</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_POST_LOT_INFO</b>
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.				
<b>ktxe_disable_exec_log</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_POST_LOT_INFO</b>
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.				
<b>ktxe_enable_doc</b>	<b>INT</b>	<b>FALSE</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROG_ARGS</b>
Flag for enabling test execution documentation data collection.				
<b>ktxe_error_gui</b>	<b>INT</b>	<b>0</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
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.				
<b>ktxe_fill_opr_dlg</b>	<b>INT</b>	<b>0</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROBER_INIT</b>
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.				

Table D-1  
Data pool items (cont.)

<b>ktxe_min_SS_touch</b>	<b>INT</b>	<b>ktxe_min_SS_touch</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_PREPARE</b>
Flag that enables/disables subsite touchdown minimalization to occur. Default state is 1, enabled. This flag is ONLY effective when the <code>ktxe_sort_subsite_ktms</code> 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 <code>UAP_WAFER_PREPARE</code> .				
<b>ktxe_missingSS_ok</b>	<b>INT</b>	<b>0</b>		
Set to non-zero to suppress the "missing Subsite error" messages. This will allow the KTD Validation step to pass without error.				
<b>ktxe_reload_wafer_plan</b>	<b>INT</b>			<b>UAP_WAFER_END</b>
Wafer plan reloading may be forced by defining this datapool item. If a <code>wwplist</code> has been modified, it will be necessary to define this item to force the next <code>wwplist</code> to be reset to the original. This item must be added at or before each <code>UAP_WAFER_END</code> . This item is removed at the end of each wafer loop.				
<b>ktxe_report_no_klf</b>	<b>INT</b>	<b>1</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_PREPARE</b>
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.				
<b>ktxe_report_no_pcf</b>	<b>INT</b>	<b>1</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_PREPARE</b>
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.				
<b>ktxe_sort_subsite_ktms</b>	<b>INT</b>	<b>ktxe_sort_subsite_ktms</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_WAFER_PREPARE</b>
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 <code>UAP_WAFER_PREPARE</code> .				
<b>last_prober_call</b>	<b>CHAR_P</b>	<b>last_prober_call</b>		
Pointer to the last prober function which encountered an error. This item is updated in <code>KTXESUP/KTXEProberErrorMessage</code> which is used by the Execution Engine. This datapool value can be used at <code>UAP_PRB_ERR_HDLR</code> .				
<b>last_prober_error</b>	<b>INT_P</b>	<b>last_prober_error</b>		
Pointer to the last prober which encountered an error. This item is updated in <code>KTXESUP/KTXEProberErrorMessage</code> which is used by the Execution Engine. This datapool value can be used at <code>UAP_PRB_ERR_HDLR</code> .				
<b>last_subsite</b>	<b>LONG_P</b>	<b>last_subsite</b>		
Pointer to the last subsite structure. Updated after subsite change. This is used to assist in getting subsite timing information. Refer to <code>KTXEAddIn:KTXEShowSubsiteTime</code> module for details.				
<b>limit_list</b>	<b>LONG_P</b>	<b>limit_list</b>	<b>UAP_VALIDATE_OCR</b>	<b>UAP_WAFER_END</b>
Pointer to the limits list returned by <code>GetLimit</code> . Updated each wafer, if the Wafer Plan file has a limits file specified.				
<b>limithashtab</b>	<b>LONG_P</b>	<b>&amp;limithashtab[0]</b>	<b>UAP_WAFER_BEGIN</b>	<b>UAP_WAFER_END</b>
Pointer to the hash table used for fast lookups of limit values.				
<p>Example:</p> <pre> LIMIT **limithashtab; LIMIT *limit_list;  limithashtab = (LIMIT**)dpGetPointer( "limithashtab" LONG_P); limit_list = (LIMIT *)limithashtab[ hash( result_id, HASHSIZE ) ]; while(limit_list != NULL) {     if (strcmp( result_id, limit_list-&gt;id ) == 0 )         break ; /* we have the limit we are looking for... */      limit_list = limit_list-&gt;nexth ; } </pre>				

Table D-1  
Data pool items (cont.)

<b>lot</b>	<b>LONG_P</b>	<b>lot</b>	<b>UAP_PROBER_INIT</b>	<b>UAP_ENGINE_EXIT</b>
Pointer to lot structure.				
<b>lotadd</b>	<b>INT_P</b>	<b>&amp;lotadd</b>	<b>UAP_PROBER_INIT</b>	<b>UAP_WRITE_LOT_INFO</b>
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: <pre>int *lotAdd ; /* set up for append to lot data */ lotAdd = ( int *)dpGetPointer( "lotadd", INT_P ) ; *lotAdd = APPENDLOT ;</pre>				
<b>lotid</b>	<b>CHAR_P</b>	<b>lot-&gt;id</b>	<b>UAP_PROBER_INIT</b>	<b>UAP_ENGINE_EXIT</b>
Lot id string.				
<b>maxErrEvtLines</b>	<b>INT_P</b>	<b>maxErrEvtLines</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROG_ARGS</b>
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: <pre>int *maxLines ; /* get pointer from data pool */ maxLines = ( int *)dpGetPointer( "maxErrEvtLines", INT_P ) ; *maxLines = newMaxLinesValue ;</pre>				
<b>maxScrollLines</b>	<b>INT_P</b>	<b>maxScrollLines</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROG_ARGS</b>
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: <pre>int *maxLines ; /* get pointer from data pool */ maxLines = ( int *)dpGetPointer( "maxScrollLines", INT_P ) ; *maxLines = newMaxLinesValue ;</pre>				
<b>next_wwp_list</b>	<b>LONG_P</b>	<b>&amp;previous_wwp_list</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
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.				
<b>next_wwp_list_ptr</b>	<b>LONG_P</b>	<b>&amp;previous_wwp_list</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
This is a pointer to the next_wwp_list pointer value. This can be adjusted to alter execution flow if desired.				
<b>previous_wwp_list</b>	<b>LONG_P</b>	<b>&amp;previous_wwp_list</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
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.				
<b>prev_wwp_list_ptr</b>	<b>LONG_P</b>	<b>&amp;previous_wwp_list</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
This is a pointer to the previous_wwp_list pointer value. This can be adjusted to alter execution flow if desired.				

Table D-1  
Data pool items (cont.)

<b>prober_wafer_id</b>	<b>CHAR_P</b>	<b>&amp;prober_wafer_id</b>	<b>UAP_VALIDATE_OCR</b>	<b>UAP_WAFER_END</b>
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:				
<pre>char *stringPtr ;  stringPtr = ( char *)dpGetPointer( "prober_wafer_id", CHAR_P ) ; printf( "old id: &lt;%s&gt;\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" ) ;</pre>				
<b>product_file</b>	<b>CHAR_P</b>	<b>&amp;product_file</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_PROBER_INIT</b>
Prober Product file string. Must be updated at or before UAP_PROBER_INIT.				
<b>result_list</b>	<b>LONG_P</b>	<b>NULL</b>	<b>UAP_TEST_END</b>	<b>UAP_HANDLE_ABORT</b>
Pointer to the results list created by a KTM.				
<b>site</b>	<b>LONG_P</b>	<b>site</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_SITE_END</b>
Pointer to the site structure. Updated before UAP_SITE_CHANGE.				
<b>sites_tested</b>	<b>INT_P</b>	<b>&amp;sites_tested</b>	<b>UAP_SITE_CHANGE</b>	<b>UAP_ENGINE_EXIT</b>
Pointer to number of sites tested. Updated before UAP_SITE_CHANGE.				
<b>skip_first_wafer_load</b>	<b>INT</b>	<b>skip_first_wafer_load</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_POST_LOT_INFO</b>
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.				
<b>skip_next_wafer_load</b>	<b>INT</b>	<b>skip_next_wafer_load</b>	<b>UAP_PRB_ERR_HDLR</b>	<b>UAP_WAFER_END</b>
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.				
<b>subsite</b>	<b>LONG_P</b>	<b>subsite</b>	<b>UAP_SUBSITE_CHANGE</b>	<b>UAP_SUBSITE_END</b>
Pointer to the subsite structure. Updated at subsite change.				
<b>sum_report_options</b>	<b>CHAR_P</b>	<b>sum_report_options</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
Summary report options from KTXE "-s" command-line switch.				
<b>total_sites</b>	<b>INT_P</b>	<b>&amp;total_sites</b>	<b>UAP_WAFER_BEGIN</b>	<b>UAP_ENGINE_EXIT</b>
Pointer to total number of sites to be tested. Updated before UAP_WAFER_BEGIN.				
<b>total_wafers</b>	<b>INT_P</b>	<b>&amp;total_wafers</b>	<b>UAP_POST_PROBER_INIT</b>	<b>UAP_POST_ENGINE_EXIT</b>
Pointer to total wafers to be tested value. Updated after UAP_PROBER_INIT.				
<b>user_arg</b>	<b>CHAR_P</b>	<b>user_arg</b>	<b>UAP_PROG_ARGS</b>	<b>UAP_ENGINE_EXIT</b>
User argument entered from "-u" command-line switch.				
<b>wafer</b>	<b>LONG_P</b>	<b>wafer</b>	<b>UAP_VALIDATE_OCR</b>	<b>UAP_WAFER_END</b>
Pointer to the wafer structure. Updated at wafer change.				

Table D-1  
Data pool items (cont.)

<b>wafers_tested</b>	<b>INT_P</b>	<b>&amp;wafers_tested</b>	<b>UAP_WAFER_BEGIN</b>	<b>UAP_WAFER_END</b>
Pointer to number of wafers tested. Updated before UAP_WAFER_BEGIN.				
<b>wdfptr</b>	<b>LONG_P</b>	<b>wdfptr</b>	<b>UAP_POST_PROBER_INT</b>	<b>UAP_POST_ENGINE_EXIT</b>
Pointer to the wdfptr, or wafer description structure. Updated before UAP_POST_PROBER_INIT.				
<b>wpf_info</b>	<b>LONG_P</b>	<b>wpf_info</b>	<b>UAP_VALIDATE_OCR</b>	<b>UAP_ENGINE_EXIT</b>
Pointer to the wafer plan structure. Used by the Analysis Software.				
<b>wwp_list</b>	<b>LONG_P</b>	<b>wwp_list</b>	<b>UAP_WAFER_BEGIN</b>	<b>last UAP_SITE_END</b>
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

The user has the ability to manipulate the data pool in the KULT modules using the data pool functions. The user must be careful about using these functions because the improper use may cause fatal errors. The list of functions that follow allow the user 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:

```
voiddpRemoveData(char *name, int type);
```

Printing items in the data pool:

```
voiddpPrintData(char *name, int type);
voiddpPrintAllData(void);
```

## Description of the data pool functions

**int dpAddData (char \*name, int type, ...);**

This function is used to add new non-pointer type data to the data pool. If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

PARAMETERS: This function takes three parameters.

**name** — character string containing the name of the data.

**type** — non-pointer type of data (INT, FLOAT, LONG, DOUBLE, CHAR).

**value** — absolute data value.

**NOTE**      *The value is interpreted internally based on the type of data.*

Return Value: This function returns one of the following statuses:

**OK\_dpAdd** — The add was successful.

**FAILED\_dpAdd** — There was a problem allocating memory for the data node. The add was unsuccessful.

Examples:

```
status = dpAddData("MyIntData", INT, 10);
```

```
status = dpAddData("MyFloatData", FLOAT, 10.11);
```

**int dpAddPointer(char \*name, int type, void\*valuep);**

This function is used to add pointer-type data to the data pool. If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

PARAMETERS: This function takes three parameters.

**name** — character string containing the name of the data.

**type** — pointer type of data (INT\_P, FLOAT\_P, LONG\_P, DOUBLE\_P, CHAR\_P).

**valuep** — value pointer.

When using a dpAddPointer call, the data value being added to the data pool must be either static or malloc'd. Do NOT use automatic pointers.

Return Value: This function returns one of the following statuses:

**OK\_dpAdd** — The add was successful.

**FAILED\_dpAdd** — There was a problem allocating memory for the data node. The add was unsuccessful.

Example: `status = dpAddPointer("MyPointerData", FLOAT_P, floatptr);`

**int dpAddArray(char \*name, int type, void\*valuep, int elements);**

This function is used to add arrays to the data pool. If data of the same name already exists in the data pool, the type and the value is overwritten with the new data.

PARAMETERS: This function takes four parameters.

**name** — a character string containing the name of the data.

**type** — array type of data (INT\_ARRAY, FLOAT\_ARRAY, DOUBLE\_ARRAY).

**valuep** — pointer to the first element of the array.

**elements** — number of elements in the array.

Return Value: This function returns one of the following statuses:

**OK\_dpAdd** — The add was successful.



**FAILED\_dpAdd** — There was a problem allocating memory for the data node. The add was unsuccessful.

Example: `status = dpAddArray("MyArrayData", INT_ARRAY, arrayptr, 10);`

**void \*dpGetDataPtr(char \*name, int type)**

This function is used to get a pointer to a value of non-pointer type data (i.e., INT, FLOAT, DOUBLE, LONG, CHAR) in the data pool. This function returns a void pointer. It is the user's responsibility to typecast the value to the appropriate data type.

PARAMETERS: This function takes two parameters.

**name** — character string containing the name of the data to be extracted from the data pool.

**type** — the data type of the variable.

Return Value:

- If the data of a specified name and type is found in the data pool, a void pointer to the value is returned. It is the user's responsibility to typecast the data to the appropriate data type (i.e. int \*, float \*, double \*, long \*, char \*).
- If the data is not found in the data pool, NULL is returned.

Example: `valueptr = (int *)dpGetDataPtr("MyInt Data", INT);`

**void \*dpGetPointer(char \*name, int type)**

This function is used to get the pointer value of pointer type data (i.e. INT\_P, FLOAT\_P, DOUBLE\_P, LONG\_P, CHAR\_P) in the data pool. This function returns a void pointer. It is the user's responsibility to typecast the value to the appropriate type.

PARAMETERS: This function takes two parameters.

**name** — character string containing the name of the data to be extracted from the data pool.

**type** — data type of the variable.

Return Value:

- If the data of a specified name is found in the data pool, a void pointer to the value is returned. It is the user's responsibility to typecast the data to the appropriate type (i.e. int\*, float\*, double\*, long\*, char\*).
- If the data is not found in the data pool, NULL is returned.

Example: `valueptr = (float *)dpGetPointer("MyPointer Data", FLOAT_P);`

**void \*dpGetArrayElement(char \*arrname, int type, int element)**

This function is used to get a specific element of an array from the data pool.

PARAMETERS: This function takes three parameters.

**arrname** — character string containing the name of the array.

**type** — data type of the variable (i.e., INT\_ARRAY, FLOAT\_ARRAY, DOUBLE\_ARRAY).

**element** — integer value specifying the index of the array.

Return Value: Returns a pointer to the specific element in the array.

- If the specified array name is found in the data pool, a void pointer to the specified element in the array is returned. It is the user's responsibility to typecast the data to the appropriate type.
- NULL is returned in one of the following situations:
- If the array name is not found in the data pool.
- If the data requested is not of an array type (i.e. INT\_ARRAY, FLOAT\_ARRAY, DOUBLE\_ARRAY).
- If the element requested is less than 0 or larger than the number of elements in the array.

Example: `valueptr = (int *)dpGetArrayElement("MyArray Data", INT_ARRAY, 7);`

### **void dpRemoveData(char \*name, int type)**

This function is used to remove specific data from the data pool. It will free up the allocated memory for the node in the data pool and the memory for the value of any of the non-pointer type data (i.e., INT, FLOAT, LONG, DOUBLE, CHAR). The user is responsible for reallocating memory for all of the pointer type data and the arrays.

PARAMETERS: This function takes two parameters.

**name** — character string containing the name of the data to be removed from the data pool.

**type** — the data type of the variable.

Return Value: This function does not return anything.

**CAUTION** Do not remove any variables put into the data pool by KTXE. This may cause fatal errors.

Examples:

```
dpRemoveData("MyArrayData", INT_ARRAY);
```

```
dpRemoveData("MyPointerData", FLOAT_P);
```

### **void dpPrintData(char \*name, int type)**

This function allows the user to print a variable and its value in the data pool by passing the name and type of data.

**NOTE** All the *dpPrint* routines print to the location specified in the *KI\_KTXE\_DEBUG\_LOG* environment variable. To print to the screen, set this environment variable to *"/dev/tty"*.

PARAMETERS: This function takes two parameters.

**name** — character string containing the name of the data to be printed.

**type** — data type of the variable.

Return Value: This function does not return anything.

Example: `dpPrintData("MyIntData", INT);`

### **void dpPrintAllData(void)**

This function allows the user to print all the data in the data pool.

**NOTE** All the *dpPrint* routines print to the location specified in the *KI\_KTXE\_DEBUG\_LOG* environment variable. To print to the screen, set this environment variable to *"/dev/tty"*.

PARAMETERS: No parameters are required for this function.

Return Value: This function does not return anything.

Example: `dpPrintAllData();`

## Recommendations, hints, and examples

The following are some recommendations and hints about using the data pool:

- Try to avoid using the `dpAdd...` and the `dpRemove...` routines. Remember that when you define a variable in the global data file, it is automatically added to the data pool.
- When using the `dpAdd...` functions, make sure the variable of that name does not already exist in the data pool. If a variable of the same name is already in the data pool, it will be overwritten by the new value you passed in with the `dpAdd` functions.
- When you use the `dpPrint...` functions, make sure the environment variable "KI\_KTXE\_DEBUG\_LOG" is set to the appropriate file location where you want the information logged.
- Do not use the `dpAdd...` or `dpRemove...` functions on any of the variables that KTXE puts into the data pool. This may cause fatal errors.

### Example:

`dpAddData/dpGetDataPtr` example:

```
{
  /* create/initialize value in datapool
   */
  int value ;

  value = 4 ;
  dpAddData( "nameOfElement", INT, value ) ;
}

{
  /* get value from datapool
   */
  int value ;

  value = *(( int *)dpGetDataPointer( "nameOfElement", INT )) ;

  /* value would now equal 4
   */
}
```

`dpAddPointer/dpGetPointer` examples:

```
/* create/initialize value in datapool
 */
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 datapool
   */
  double *dblPtr ;
  double dbl ;

  dblPtr = ( double *)dpGetPointer( "myPtrName", DOUBLE_P ) ;
  dbl = *dblPtr ;
  /* dbl = 5.6 now */

  /* Modify value. NOTE that we do not have to "refresh" the
   * datapool. Since this is a pointer datapool value
   */
  *dblPtr = 7.8 ;
}

{
  /* Read modified datapool value
   */

  double *dblPtr ;
  double dbl ;

  dblPtr = ( double *)dpGetPointer( "myPtrName", DOUBLE_P ) ;
  dbl = *dblPtr ;
  /* dbl = 7.8 now */
}

/* CHAR_P usage:
 */

{
  char *stringVar ;

  /* Create and initialize datapool 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" */
}
```

**E**

# **User Access Points (UAPs)**

---

## User Access Points (UAPs)

User Access Points (UAPs) are supported to let users 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. For example, here is a file containing one engine's UAPs:

```

/* 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,missmatch 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 desired
FAKE=YES

/*End of File */

```

## User access point usage

UAPs can access data items stored or referenced through the data pool. See paragraph on data pool items 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/or Chuck. At the UAP\_WAFER\_MISMATCH UAP the wafer positions can be queried (not all probers allow this action, please refer to the KI specific prober manual), 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 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 KI-specific prober manual). 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 KTE. For more information on SofTouch, refer to the Prober and Prober Drivers Manual.

**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 .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 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. And, 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. KUI functions may not be used to display information. Use an external program for 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 UAPs

There are two different types of UAPs:

- User Library modules, created by KULT
- .ktm files, created by KITT
  1. The format for user library modules created by KULT is:
 

```
UAP_aaaa,user_library_name,module_name(arg1,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)
```
- 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 is 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.).

2. The formats for .ktm files created by 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

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.

## Usage of LPTLIB functions at UAPs

LPTLIB functions can not be used at the following UAPs:

- UAP\_PROG\_ARGS
- UAP\_CASSETTE\_LOAD
- UAP\_LOT\_INFO
- UAP\_ENGINE\_EXIT

LPTLIB functions may not be used within user library functions or ktms called at these UAPs.

## Distributed user libraries

The following libraries and routines are available for use as reference routines for UAP code development.

- **KITTAddIn** — Legacy routines that allow the user to insert or extract data elements into or from arrays. KTE V4.2.2 and up supports array notation within 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 .gdf file from the current contents of the datapool. 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 could be modified to perform any custom processing of results that exceed a limit value. In this library, some KUI functions are included.
- **KTXEAddIn** — Routines that demonstrate access to datapool values to control 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 the user copy/rename, the desired routine to their own libraries for use within KITT and KTXE.

## Test macro debugging

Since the Datapool is only available while executing KTXE (KITT does not generate a Global Datapool), debugging your ktm in KITT is very difficult. `DBG_gdfCreate` helps solve this problem by creating a “snapshot” of the current Global Datapool while executing KTXE.

By executing the `DBG_gdfCreate` macro at a UAP just prior to the point your ktm is experiencing problems, it is possible to capture the contents of the Global Datapool 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 which will generate a valid Global Data File based on the current contents of the Global Datapool. 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 Datapool 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`, as this log file is used as temporary storage for the datapool data during the creation of the `.gdf` file.*

This function can be found in the `KI_DEBUG KULT` library.

- 
- A**
- Abort flags ..... 3-18
  - Adaptive test
    - Installation of demonstration files ..... 2-81
    - Installation of user libraries ..... 2-80
  - Adaptive testing ..... 2-71
  - AddNew ..... A-23
  - Addtester ..... 5-16
  - Advanced data pool use ..... D-11
  - Array Support ..... 2-32
- C**
- Cassette plan builder ..... 2-64
  - Cassette plan builder file menu ..... 2-65
  - Cassette plan builder help menu ..... 2-66
  - Cassette plan builder options menu ..... 2-66
  - Cassette test plan file format ..... C-18
  - Command line interface ..... 2-49
  - Configuring the tool.tpi file ..... 5-18
  - ContSkipAbortDlg — Continue Skip Abort Message
    - Dialog ..... B-14
  - Copying user libraries with kult\_copy\_lib ..... 2-50
  - CreateNew ..... A-23
  - Creating Test Plans ..... 2-1
- D**
- Data Analysis ..... 4-1
  - Data logging routines ..... A-11
  - Data output formats ..... 4-2
  - Data Pool ..... D-1
  - Data pool ..... D-2
  - Data window ..... 4-19
  - DeleteLimit ..... A-21
  - DeleteLimitCode ..... A-20
  - DeleteLot ..... A-19
  - DeleteParam ..... A-20
  - DeleteSite ..... A-20
  - DeleteWafer ..... A-19
  - Description of the data pool functions ..... D-12
  - Description window ..... 2-47
  - Distributed user libraries ..... E-5
- E**
- Edit-time library locking ..... 2-54
  - EndLot ..... A-14
  - EndSite ..... A-15
  - EndWafer ..... A-14
  - Environment variables ..... 5-7
  - Error and event logging ..... 3-16
  - Execution engine selection field ..... 2-66
  - Execution process ..... 3-8
- F**
- File description field ..... 2-66
  - File management ..... 5-11
  - FileExist ..... A-18
  - FindFirst ..... A-24
  - FindLast ..... A-25
  - FindNext ..... A-25
  - FindPrev ..... A-26
- G**
- Generating documentation during testing ..... 2-71
  - GetComment ..... A-21
  - GetLimit ..... A-22
  - GetLimitCode ..... A-21
  - GetLot ..... A-15
  - GetLotData ..... A-18
  - GetParam ..... A-17
  - GetParamList ..... A-17
  - GetProgramArgs — Get Program Command Line
    - Arguments ..... B-4
  - GetSite ..... A-16
  - GetStartTime ..... A-19
  - GetWafer ..... A-15
  - Global data file format ..... C-11
  - Global data file selection field ..... 2-66
  - Global data variables ..... 2-73, 2-74
- H**
- Hidden libraries ..... 2-52
- I**
- InitUINew — Initialize User Interface Library ..... B-5
  - InputMsgDlg — Input Message Dialog ..... B-6
  - Insert buttons ..... 2-63
  - InsertNew ..... A-26
  - Introduction ..... 1-1, 2-2, 3-2, 4-2, 5-2, B-2
- K**
- KCAT main window ..... 4-16
  - KCM utility ..... 5-12
  - KDF ..... A-1
  - KDFtoKCS File Conversion Utility ..... 4-20
  - Keithley Curve Analysis Tool (KCAT) ..... 4-16
  - Keithley data file format ..... C-21
  - Keithley Data Files (KDF) library ..... 5-19, A-2
  - Keithley Interactive Test Tool (KITT) ..... 2-27
  - Keithley Operator Interface Editor (KOPED) ..... 3-2
  - Keithley plot file format ..... C-24
  - Keithley Summary Utility (KSU) ..... 4-2
  - Keithley Test Execution Engine (KTXE) ..... 3-4
  - Keithley test macro (.ktm) ..... C-7
  - Keithley Test Plan Manager (KTPM) ..... 2-60
  - Keithley tool palette ..... 5-2
  - Keithley User Interface ..... B-1
  - Keithley User Interface (KUI) library ..... 5-20
  - Keithley User Library Tool (KULT) ..... 2-41
  - KITT main window ..... 2-27
  - KITT Math, Array and Logic Expression Support ..... 2-31
  - KITT results window ..... 2-30
  - KOPED edit menu ..... 3-4
  - KOPED file menu ..... 3-3
  - KOPED help menu ..... 3-4
  - KOPED main window ..... 3-3
  - KOPED options menu ..... 3-4
-

- KSU description ..... 4-7
  - KSU main window ..... 4-7
  - KTE data usage ..... 2-36
  - KTE File Formats ..... C-1
  - KTE KUI localization ..... B-15
  - KTE library locking ..... 2-54
  - KTE Support Utilities ..... 2-81
  - ktpm.ini file ..... 5-6
  - KTXE main window ..... 3-5
  - KTXE results and their structures ..... 3-21
  - KTXE\_AT user library ..... 2-78
  - KTXE\_RP user library ..... 2-72
  - KTXEErrorHandler function ..... 3-15
  - KULT main window ..... 2-41
  - KULT module file format ..... C-25
- L**
- LBoxDlg — List Box Message Dialog ..... B-14
  - LFE main window ..... 2-56
  - LimitExist ..... A-28
  - Limits data entry area ..... 2-57
  - Limits editor dialog window ..... 2-59
  - Limits File Editor (LFE) ..... 2-55
  - Limits file selection field ..... 2-62
  - Linear Parametric Test Library (LPTLIB) ..... 5-19
  - Localization ..... B-15
  - Locking a module ..... 2-51
  - Log file environment variables ..... 5-10
  - Logging a Linked List of PARAMs, Data Retrieval  
using GetLotData ..... A-34
  - Logging one PARAM at a time, Data Retrieval through  
Get routines ..... A-31
  - Lot ID data field ..... 2-66
  - Lot suspend/resume ..... 3-17
  - LotDlg — Lot Information Dialog ..... B-6
  - LotExist ..... A-19
- M**
- Main window ..... 2-67
  - Manual addenda ..... 1-3
  - Manual contents ..... 1-2
  - MatchParam2Limit ..... A-18
  - Math ..... 2-31
  - Migrating user libraries with migrate\_usrlib ..... 2-51
  - Multi-cassette Multi-lot Testing Utility (multi\_cassette)  
3-17
- N**
- New site plan button ..... 2-64
- O**
- OkCancelAbortMsgDlg — Ok Cancel Abort Message  
Dialog ..... B-7
  - OkCancelMsgDlg — Ok Cancel Message Dialog ..... B-7
  - OkMsgDlg — Ok Message Dialog ..... B-7
  - Other documentation ..... 1-3
  - Other KSU main window controls ..... 4-14
  - Overview of the manual ..... 1-2
- P**
- Parameter limits file format ..... C-14
  - Parameter Set Editor (PSE) ..... 2-38
  - Parameter set file format ..... C-12
  - Parameter window ..... 2-45
  - Plan type selection button ..... 2-63
  - Pop-up menus ..... 3-4
  - Pop-up tools menu ..... 4-19
  - Preview ..... 4-2
  - Probe card file format ..... C-10
  - Probe card file selection field ..... 2-63, 2-66
  - Probe Pattern Editor window ..... 2-9
  - Probe patterns/Site plans list ..... 2-63
  - Programming examples ..... A-6
  - Project environments ..... 5-12
  - PSE main window ..... 2-38
  - PutComment ..... A-21
  - PutLimit ..... A-22
  - PutLot ..... A-11
  - PutParam ..... A-13
  - PutParamList ..... A-13
  - PutSite ..... A-12
  - PutWafer ..... A-12
- Q**
- QuitUI — Quit User Interface ..... B-7
- R**
- Random Pattern Generation (rand\_pat) ..... 2-81
  - Raw report ..... 4-6
  - Recommendations, hints, and examples ..... D-15
  - Remove ..... A-27
  - Result based testing ..... 2-73
  - Run-time library locking ..... 2-54
- S**
- S530 environment variables ..... 5-15
  - Sample display ..... 5-15
  - Sample programs ..... A-31
  - Scaling window ..... 4-18
  - ScrollMsgDlg — Scrollable Message Dialog ..... B-7
  - ScrollMsgDlgClr — Scrollable Message Dialog Clear  
B-8
  - ScrollMsgDlgMsg — Scrollable Message Dialog  
Message ..... B-8
  - Select tester ..... 5-14
  - Setup Information ..... 5-14
  - Site Editor window ..... 2-12
  - Site Optimization window ..... 2-14
  - Site plan/Wafer plan builder field ..... 2-63
  - Slot ID type selection buttons ..... 2-67
  - Sort subsites button ..... 2-63
  - Standard report ..... 4-4
  - Standard UAP file field ..... 2-67
  - Starting KSU ..... 4-7
  - StatusDlg — Status Dialog ..... B-8
  - Structure definitions ..... A-31
  - Structure handling routines ..... A-23
  - Summary reports ..... 4-3
  - System Administration ..... 5-1
  - System configuration ..... 5-4
  - System customization ..... 5-17
  - System integration ..... 5-19
  - System software ..... 1-3
- T**
- Target Setup window ..... 2-8

Test data logging .....A-2  
 Test documentation tool ..... 2-67  
 Test Execution ..... 3-1  
 Test macro debugging .....E-6  
 Test macros/Site plans list ..... 2-63  
 Test Structure File Editor (TSE) ..... 2-23  
 Test structure file format ..... C-6  
 transfer test plans ..... 5-12  
 Troubleshooting ..... 2-55  
 TSE main window ..... 2-23  
 Types of UAPs .....E-4

**U**

UAP files ..... 2-73  
 UAP locations .....E-5  
 UAP module/KTM area ..... 2-67  
 Update comment routines .....A-21  
 Update limits routines .....A-21  
 UpdateModelessDlg — Update Modeless Dialogs ..  
     B-9  
 UpdateStatusDlg — Update Status Dialog .....B-10  
 Usage of LPTLIB functions at UAPs .....E-5  
 Use of ibupu in KITT ..... 2-38  
 User access point file format ..... C-23  
 User access point usage .....E-3  
 User Access Points (UAPs) ..... E-1, E-2  
 User interface constants .....B-2  
 User interface library functions .....B-4  
 User interface library variables .....B-3  
 User Library files required for KTXE execution ..3-15  
 Using Generated Identifiers in a Test Macro ..... 2-35  
 Using limits files .....A-3  
 Using the test documentation tool ..... 2-69

**V**

VarMsgDlg — Variable Message Dialog .....B-10

**W**

Wafer description file description field ..... 2-64  
 Wafer description file format ..... C-3  
 Wafer description file selection field ..... 2-62, 2-66  
 Wafer Description Utility (WDU) ..... 2-3  
 Wafer Graph Editor window ..... 2-14  
 Wafer id usage in KTXE ..... 3-17  
 Wafer plan builder ..... 2-60  
 Wafer plan builder file menu ..... 2-61  
 Wafer plan builder help menu ..... 2-62  
 Wafer plan builder main window ..... 2-60  
 Wafer plan builder options menu ..... 2-62  
 Wafer plan file description field ..... 2-64  
 Wafer Setup window ..... 2-6  
 Wafer test plan field ..... 2-67  
 Wafer test plan file format ..... C-16  
 Warranty information ..... 1-3  
 WDU main window ..... 2-3  
 WfrIdDlg — Single Wafer Information Dialog ....B-12  
 WfrIdsDlg — Multiple Wafer Information Dialog .B-11  
 Workstation applications ..... 5-2

**Y**

YesNoAbortMsgDlg — Yes No Abort Message Dialog  
     B-13  
 YesNoCancelMsgDlg — Yes No Cancel Message  
 Dialog .....B-14

**Z**

Zone based testing ..... 2-71  
 Modes ..... 2-71



Specifications are subject to change without notice.  
All Keithley trademarks and trade names are the property of Keithley Instruments, Inc.  
All other trademarks and trade names are the property of their respective companies.

**Keithley Instruments, Inc.**

Corporate Headquarters • 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 • 1-888-KEITHLEY • [www.keithley.com](http://www.keithley.com)

---

**KEITHLEY**

A Tektronix Company

A Greater Measure of Confidence