Batch Commands User’s and Reference Manual

Support for

Calibre® DESIGNrev™
Calibre® WORKbench™
Calibre® LITHOview™
Calibre® MDPview™

Software Version 2006.4

Table of Contents

List of Figures

List of Tables

Chapter 1
Conventions Used In This Manual

As you work through this material, you will use three types of commands:

- *Basic Tcl Commands*

  The core commands for the Tcl language. These commands work in any Tcl program. Within this document, all basic Tcl Commands are displayed in **orange.**

- *Object Class Commands*

  Commands for creating and manipulating application-specific objects. Many of the object class commands used within this tutorial are Mentor Graphics extensions to the Tcl language, and can only be used when running Calibre WORKbench. Within this document, object class commands are displayed in blue. They are active links and take you to the reference pages containing complete command syntax.

- *Instance Commands*

  Commands for working with a specific instance of an object, its contents, and its properties. These only work if you have first created the instance using one of the object class commands. The instance commands are part of the object class definitions, so using them requires that you have access to the same extensions used when creating the instances. Within this document, instance commands are displayed in blue. They are active links and take you to the reference pages containing complete command syntax.

Reference Syntax

Within this document, the following notational elements distinguish between those arguments that are required or optional, user-supplied or keyword.

Bold	A bold font indicates a required argument
[ ]	Square brackets enclose optional arguments. Note that for Tcl syntax examples, some brackets represent evaluated expressions.
Italic	An italic font indicates a user-supplied argument.
{ }	Braces enclose arguments to show grouping. In certain cases, braces are actual syntactical objects, required by an operation or statement. When this is the case, the documentation explicitly states that the braces are required.
|	A vertical bar indicates an either/or choice between items.

…	Braces that include ellipses indicate that the enclosed argument(s) may be repeated one or more times.
2	A superscript value outside square brackets indicates that the enclosed argument can be repeated up to that many times.
*	An asterisk outside square brackets indicates that the enclosed argument is optional and can be repeated as many times as needed.

Note

This document introduces you to some of the Tcl and Tk basics you will need to know to use the batch commands for Calibre WORKbench, DESIGNrev, and MDPview. It is not designed to teach Tcl/Tk. Rather, it is designed as a starting point, introducing topics you may want to explore more deeply through the classes, books, and websites available elsewhere.

Topics Covered in This Manual

Chapter 2
Working in the Tcl Environment

The batch commands for Calibre WORKbench, DESIGNrev, and MDPview are proprietary Tcl-based commands providing access to internal application functions. You may ask, “Why Tcl?”

- Tcl/Tk is easy to learn. In fact, Tcl/Tk is considered to be one of the easiest programming languages to learn.
- Tcl/Tk is easy to customize. The ability to customize makes it possible for Mentor Graphics Corporation to provide you with many proprietary objects and commands that simplify programming.
- Tcl/Tk is open source. Many people have developed Tcl extensions and Tk widgets and made them publicly available to you.
- Tcl/Tk is a tool control language. This means it lets you send data from one program to another.

Mentor Graphics Corporation recommends that you take advantage of one or more of the excellent books and websites on the language. The list that follows is provided to give you a place to start in your search for the reference material that works best for you. It is not an endorsement of any book or website. While every attempt has been made to ensure that the URLs listed here point to active websites, inclusion here does not guarantee this to be so.

Books

Practical Programming in Tcl and Tk (4th Edition)
Brent B. Welch, Jeffrey Hobbs, Brent Welch
Prentice Hall PTR (2003)
Tcl/Tk in a Nutshell
Paul Raines, Jeff Tranter
O’Reilly and Associates, Inc. (1999)

Tcl and the Tk Toolkit
John K. Ousterhout
Addison-Wesley Professional (1994)

Tcl/Tk Tools
Mark Harrison
O'reilly (1997)

Websites

Tcl Developer Xchange —
http://www.tcl.tk/scripting/

TclTutor —
http://www.msen.com/~clif/TclTutor.html

Beginning Tcl — http://wiki.tcl.tk/298

The Tcler’s Wiki — http://wiki.tcl.tk

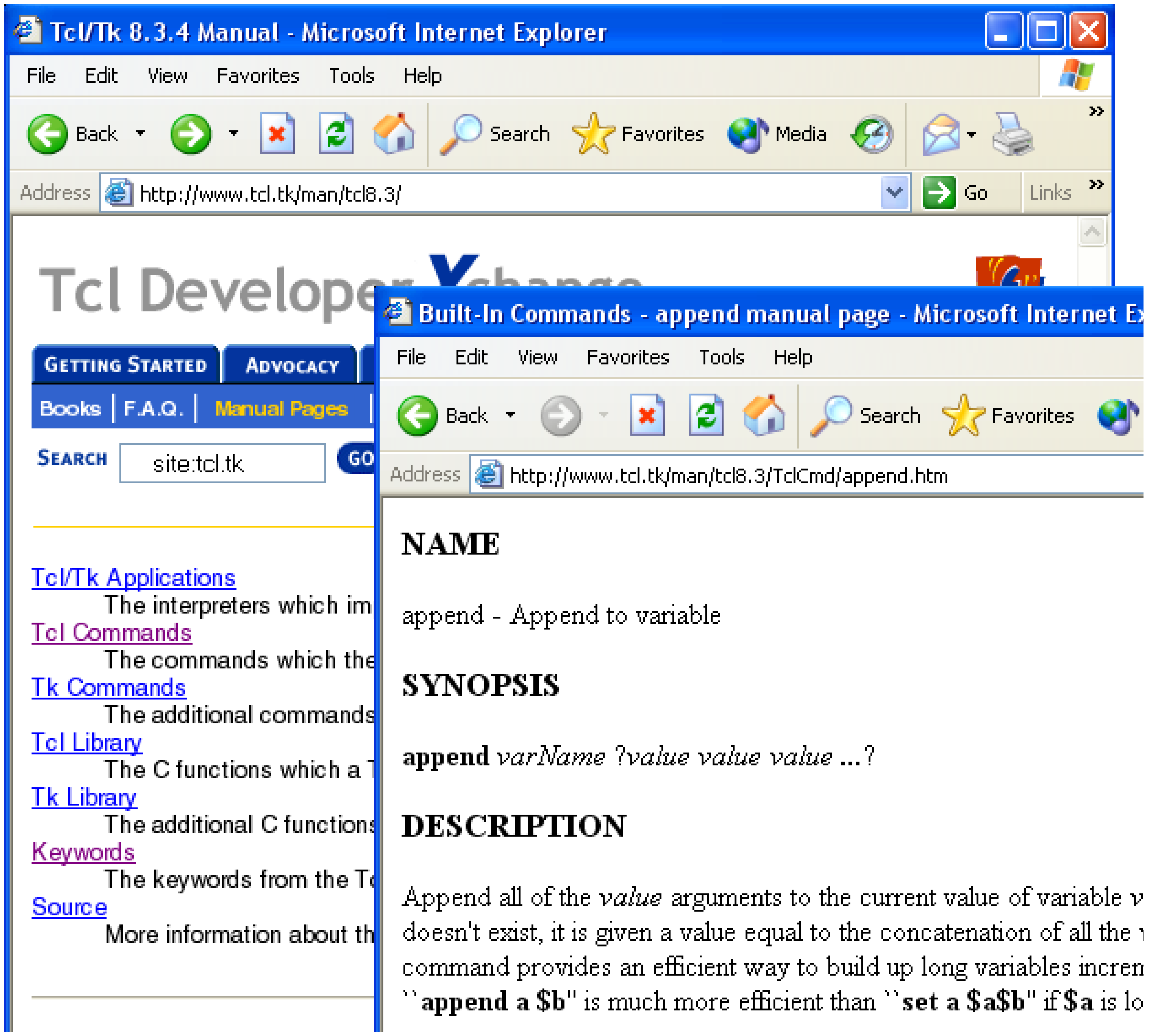
ActiveState.com —
http://www.activestate.com/Products/ActiveTcl/

Getting Help on Tcl

As you develop scripts and macros with Tcl/Tk, you may find it helpful to check the syntax of the various commands you want to use.

- Access manpages for Tcl and Tk commands at the Tcl Developers Exchange: http://www.tcl.tk/man/tcl8.4/ (upgraded from 8.3)
- Find reference pages for the batch commands in "Batch Commands for Layout Manipulation" on page 89 of this manual.

Figure 2-1. Manpages Available on the Internet


Tcl/Tk 8.3.4 Manual - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://www.tcl.tk/man/tcl8.3/ Go Links
Tcl Develope
GETTING STARTED ADVOCACY
Books F.A.Q. Manual Pages
SEARCH site:tcl.tk GO
Tcl/Tk Applications
The interpreters which im
Tcl Commands
The commands which the
Tk Commands
The additional commands
Tcl Library
The C functions which a T
Tk Library
The additional C function
Keywords
The keywords from the T
Source
More information about th
Built-In Commands - append manual page - Microsoft Internet E
File Edit View Favorites Tools Help
Back Search Favorites
Address http://www.tcl.tk/man/tcl8.3/TclCmd/append.htm

NAME

append - Append to variable

SYNOPSIS

append *varName* ?*value value value ...*?

DESCRIPTION

Append all of the *value* arguments to the current value of variable *v*
doesn't exist, it is given a value equal to the concatenation of all the
command provides an efficient way to build up long variables incren
``**append a $b**'' is much more efficient than ``**set a ab**'' if **$a** is lo

Tcl Versus Tk and What the Distinction Means to You

As you read about Tcl you will find it is rarely mentioned without Tk. This reflects the close relationship between the two:

> ***Tcl*** is the ***language*** used to write the scripts.
>
> ***Tk*** is the ***toolkit*** of building blocks used to build graphical user interfaces for Tcl programs. It is an extension to Tcl.

Many people who write scripts using the batch commands for Calibre WORKbench, DESIGNrev, and MDPview never bother to create a graphical user interface for their scripts. If you are one of those people, you will not need to learn Tk. However, you do need to understand a few things about how Tcl and Tk fit together.

- To use either Tcl or Tk, your environment must be set up to give you access the proper binary files, libraries, and necessary extensions.
- You can use Tcl without Tk.
- You cannot use Tk without Tcl. Hence, you will often see the Tk toolkit written as Tcl/Tk.
- Any time you run one of the tools (Calibre WORKbench, DESIGNrev, or MDPview), that tool loads everything you need to successfully execute Tcl commands.
- If you run one of the tools in any of the graphical user interface modes, that tool also loads everything you need to build a graphical user interface with Tk.

Note

If you are not running the tools in a GUI mode, you cannot use Tk. If the tool attempts to execute a Tk command when the Tk software package is not loaded, it aborts with an error.

.signaext file is a script written in the Tcl programming language. It can contain any valid Tcl and Tk commands. Calibre tools automatically ‘source’ (load and run the commands in) .signaext files that exist in the user’s home directory and the current directory.

Because many .signaext files contain Tk commands[1] (though you may not be aware of it) and because the tools process the .signaext file every time they invoke, many users experience difficulties when they attempt to invoke Calibre WORKbench, DESIGNrev, or MDPview in ***-shell*** or any of the other non-GUI modes. You can avoid this sort of difficulty by embedding the Tk commands within conditional statements that instruct the tool to execute them only if the Tk software package is loaded.

1. Setting GUI-related variables and creating macros both require use of Tk commands.

For example:

# unprotected Tk commands	**# protected Tk commands**

#--------------------------
 set filename [tk_getSaveFile]
	#--------------------------
	 if [isTkLoaded] {
	 set filename [tk_getSaveFile]
	 }

Extensions to Tcl

One of the reasons Tcl/Tk is used so widely is that is relatively easy to create extensions to the language. Tk was the first extension created. Many other publicly available extensions have been created since. Calibre WORKbench, DESIGNrev, and MDPview use two:

- **iTcl/iTk**, also called Incr Tcl, is an extension to Tcl that provides object-oriented programming capabilities. You can learn more about iTcl/itk at http://incrtcl.sourceforge.net/itcl/
- **BLT** is an extension to Tcl/Tk that provides a set of widgets useful for displaying and manipulating data. It includes graph and bar chart widgets, drag and drop objects, and other fairly advanced GUI objects. You can learn more about BLT at http://wiki.tcl.tk/199

Because Calibre WORKbench, DESIGNrev, and MDPview use these extensions, the extensions are available for use in any scripts that you write to run with these applications.

These applications also rely on proprietary extensions created by Mentor Graphics Corporation. Mentor Graphics extensions provide you with the commands and objects that manage layout databases in memory. These extensions are described in the reference chapters of this manual. Some of the proprietary commands are not dependant on Tk, and so can be run in any mode. Others require that Tk be available and can only be in one of the GUI modes. Refer to the individual reference pages to discover the requirements of each command.

Things You Must Know About Tcl

Tcl is Case Sensitive

In Tcl/Tk, “layout” and “Layout” are two different strings. When you type in a Tcl/Tk command or variable, make sure you type the case correctly.

The core set of Tcl/Tk commands are all lower case, so when in doubt, use lower case. However, when people write extensions to Tcl/Tk, they can choose whether or not to capitalize the new commands. The reference pages in Part “Batch Commands for Layout Manipulation” on page 89, display command syntax in the proper case.

Many Characters Have Special Meanings in Tcl

In Tcl/Tk scripts, you will often see characters used for special purposes. For a complete list of special characters, you should consult the Tcl/Tk resources listed at the beginning of this chapter. The list that follows describes the special characters you will encounter when reading the examples in this manual:

; — The semicolon terminates the previous command, allowing you to place more than one command on the same line.

\ — Used alone, the backslash continues a command on the following line.

\$ — The backslash with other special characters, like a dollar sign, instructs the Tcl interpreter to treat the character literally.

\n — The backslash with the letter “n” instructs the Tcl interpreter to create a new line.

$ — The dollar sign in front of a variable name instructs the Tcl interpreter to access the value stored in the variable.

[ ] — Square brackets group a command and its arguments, instructing the Tcl interpreter to treat everything within the brackets as a single syntactical object. You use square brackets to write nested commands.

Example: To set a variable to the result of processing a command:

```
set my_layout [layout create sample.gds]
```

{ } — Curly braces instruct the Tcl interpreter to treat the enclosed words as a single string. The Tcl interpreter accepts the string as is, without performing any variable substitution or evaluation.

Example: Create a string that contains special characters, such as $ or \:

```
set my_string {This book costs $25.98.}
```

" " — Quotes instruct the Tcl interpreter to treat the enclosed words as a single string. However, when the Tcl interpreter encounters variables or commands within string in quotes, it evaluates the variables and commands to generate a string.

Example: Create a string that displays a final cost calculated by adding two numbers:

```
set my_string “This book costs \$[expr $price + $tax]”
```

Tcl Comments Can Be Tricky

In Tcl/Tk, you create the equivalent of comments with the pound sign “#”. The pound sign directs the Tcl compiler to not evaluate the rest of the line.

Tricky Point 1: Evaluate does not equal parse. Despite the pound sign, the comment below gives an error because Tcl detects an open lexical clause.

```
# if (some condition) {
if { new text condition } {
...
}
```

Tricky Point 2: The apparent beginning of a line is not always the beginning of a command:

```
# This is a comment
```

In the following code snippet, the line beginning with “# -type” is not a comment, because the line right above it has a line continuation character (\). In fact, the “#” confuses the Tcl interpreter, resulting in an error when it attempts to create the tk_messageBox.

```
tk_messageBox -message "The layout hier was successfully written."\
# -type ok

# and this is also a comment. This one will span \
multiple lines, even without the # at the beginning \
of the second and third lines.
```

In general, it is good practice to begin all comment lines with a #.

Tricky Point 3: The beginning of a command is not always at the beginning of a line:

Usually, you begin new commands at the beginning of a line. That is, the first character that is not a space will be the first character of the command name. However, you can combine multiple commands into one line using the semicolon “;” to signal the end of the previous command:

```
set myname “John Doe” ;set this_string “next command”

set yourname “Ted Smith” ; #this is a comment
```

Tcl Objects Have Handles

A *handle* is a system-generated instance name by which Tcl identifies an object it creates in memory. When you create a Tcl object using the appropriate Tcl command, the command generates the object's handle, and returns it to you as the output (or return value) of the command. You use these instance names to refer to Tcl objects within a command or script.

The distinction here is between *handles* (system-generated instance names) and *names* (user-defined names). The reason for making this distinction is to call attention to the differences between the following:

- instances of objects versus classes of objects:
    - *layout* is a class of objects. It is a generic name that encompasses all layouts.
    - *layout0* and *layout1* are handles for instances of a layout. Each is a system-generated name that applies to one and only one specific layout.
- names the program gives to instances versus names by which you refer to them. For example, the command "`layout create mylayout.gds`" creates all of the following items:
    - *layout is the type of object you are creating.*
    - *layout0* is the system generated name (handle) for the layout in memory.
    - *mylayout.gds* is the name of the layout file.
    - *mylayout* is the name by which you generally refer to the layout.
- names that are used as part of a command name versus those that are not
    - *T*he Object Class Commands that create and manipulate layout objects all begin with the string "*layout*".
    - The Instance Commands that manipulate the contents of a specific layout all begin with the layout's handle: for example, "`layout0 create cell topcell`".
    - *No commands begin with the string "topcell"*. You manipulate the top-level cell called "topcell" through various instance commands, named according to the name of the instance containing topcell.

Most Database Objects are not Tcl Objects

All Tcl objects have handles. The following are Mentor Graphics Tcl objects:

- cwbWorkBench
- layout
- layoutView

Many database objects are not Tcl objects. Within memory, they are actually stored as elements of Tcl objects. The following are *not* Tcl objects:

- Layers
- Cells
- Polygons
- Text
- Paths
- Vertices
- Srefs (single cell references)
- Arefs (arrays of cell references)

You manipulate a database object that is not a Tcl object through the various instance commands for the layout to which the object belongs. Thus, for most of the work you do on the contents of a layout, you use the $L commands.

Questions and Answers

Q: Where do I get handles?

A: When you create an object, the command returns the handle of the newly created object. In addition, many other commands return handles. For example, the cget method for the cwbWorkBench class objects can return either layout handles or layoutView handles.

Q: How do I reference objects that do not have handles?

A: For objects that do not have handles, you must supply enough of a description of that object to uniquely identify it. For example:

To delete a polygon from a layout, you must supply:

- Layout handle, cell name, layer, coordinates of all vertices

To delete a cell from a layout, you must supply:

- Layout handle, cell name

The descriptions of the commands for manipulating non-Tcl objects tell exactly what information is required.

Q: Is there anything I can use besides the handle, like a name, to reference a Tcl object?

A: Yes, you can save handles to variables, and use the variable instead of the handle. It is a good (almost necessary) practice to save the handles as variables for any object you may need to manipulate. While handles and variable names are both strings, variables are easier to work with because you decide what they are named. Handles are system generated, so you have no control over what a particular object's handle will be. Variables are also reusable, and context independent.

Q: I have a viewer open and I want to issue commands that require that I know the layout handle. How can I get it?

A: The following code snippet will save the layout handle in the variable “layout”. Note that this only works if you only have one viewer open. If you have more than one layout loaded into the viewer, it returns the handle of the current (visible) layout.

```
% set wb [find objects -class cwbWorkBench]
```

```
        % set layout [$wb cget -_layout]
```

Q: Are there any built-in handles I can use?

A: Yes. And you can try using them in a pinch. In general, it is much better to capture handles as the application returns them, (assigning them to a variable) as this guarantees that you'll be pointing to the right objects.

- Windows — ::cwbWorkBench::cwbWorkBench*x* (x=0 through (windows open -1))
- Layouts — layout*n* (n=0 through (layouts loaded/created -1))
- View — ::cwbLayoutView*m* (m=0 through (windows open -1))

Assuming you can guess the handle for a Tcl object is always risky because many things cause handles to be allocated. For example, assume you attempt to create a Tcl layout object containing the data in an existing gds file. If you type the filename incorrectly, Tcl allocates the handle before it discovers it cannot open the file. When you reissue the `layout create` command, supplying the correct file name, the handle might be layout3, not layout0 as you expected.

Chapter 3
Getting Started

You can run Calibre WORKbench, DESIGNrev, and MDPview in any of four modes: Interactive GUI Mode, Interactive Shell Mode, Batch Mode, and Batch GUI Mode. Used alone, the commands **calibrewb, calibredrv, and calibremdpv** invoke the associated applications in interactive GUI mode. You must supply optional command switches to you operate in different modes.

Table 3-1. Invocation Options

	Option	**Mode**	**Description**
Interactive	-shell	interactive shell	Opens just the terminal shell window (no GUI). To interact with the program you must type commands into the shell.
	-s *script* | -startscript *script*	either interactive mode	Evaluates the specified Tcl start script on invocation. Does not work with --.
	-m *filename* [-m *filename2*]*	either interactive mode	Loads the specified layout file(s) on invocation; it accepts either GDS or OASIS files. If you are loading multiple files, each file name must have its own -m option associated with it.
	-l *layerfile*	either interactive mode	Must be used with -m. Loads a single layer properties file to be used with the filename specified in the -m option.
	-noedit	either interactive mode	Prohibits editing the layout.
	-64	any	Invokes processing in 64-bit mode.
	-h	any	Prints command usage.
	-rve [“*rveFile rveArgs*”]	interactive GUI	Runs Calibre RVE. See “Running RVE From the Command Line.”
	-b *bookmarkname*	interactive GUI	Loads a previously-created specified bookmark.
	-dl *default_file*	interactive GUI	Specifies a default layer properties file.

Table 3-1. Invocation Options

	Option	**Mode**	**Description**
Batch	--	any	Designates that all remaining arguments are passed directly to a script's argv variable without interpreting them. Does not work with -s.
	-a	batch	Evaluates a single batch command.
	script_name	batch	Evaluates the specified Tcl script and exits.
	-gui	batch GUI	Evaluates the script using the functions provided by the Tk toolkit. Exits after processing.
	-64	any	Invokes processing in 64-bit mode.

Issuing Tcl Commands in Interactive Mode

The Interactive mode allows you to interact with the application directly and dynamically, issuing commands to load, examine, and manipulate data. Your choices are:

- Interactive GUI Mode
- Interactive Shell Mode

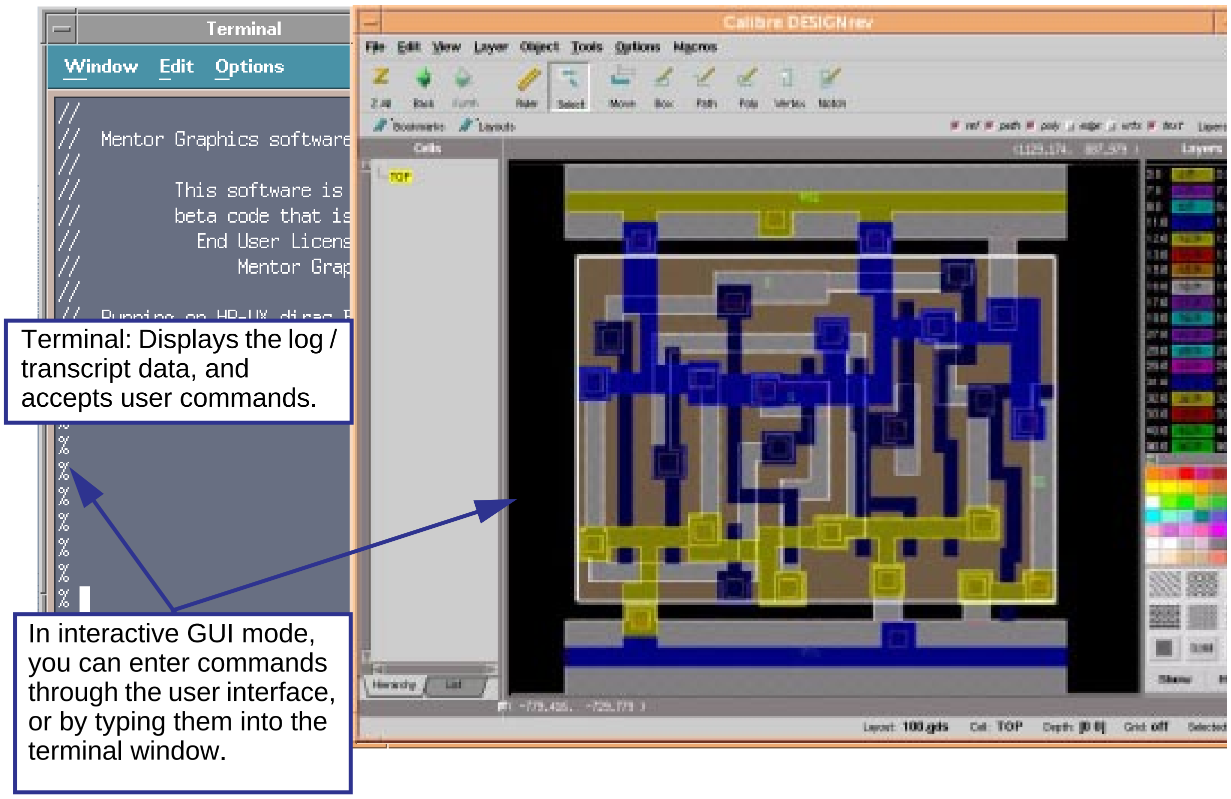
Interactive GUI Mode

When you invoke in the interactive GUI mode, you have access to both the viewer and the shell (also called terminal) window. In the viewer, you issue commands using the mouse, menus, and dialog boxes. In the shell window, you issue commands by typing them.

You invoke in interactive GUI mode whenever you issue the **basic** command[1] without a script name or the ***-a*** or ***-shell*** arguments.

1. calibrewb, calibredrv, calibrelv, or calibremdpv

Figure 3-1. Interactive GUI Mode



```
//
//  Mentor Graphics software
//
//         This software is
//         beta code that is
//           End User Licens
//              Mentor Grap
//
//  Running on HP-UX dirac
%
%
%
%
%
%
%
```

Terminal: Displays the log / transcript data, and accepts user commands.

In interactive GUI mode, you can enter commands through the user interface, or by typing them into the terminal window.

Caution

Do not start the application as a background process. The Tcl shell will lock up.

Running RVE From the Command Line

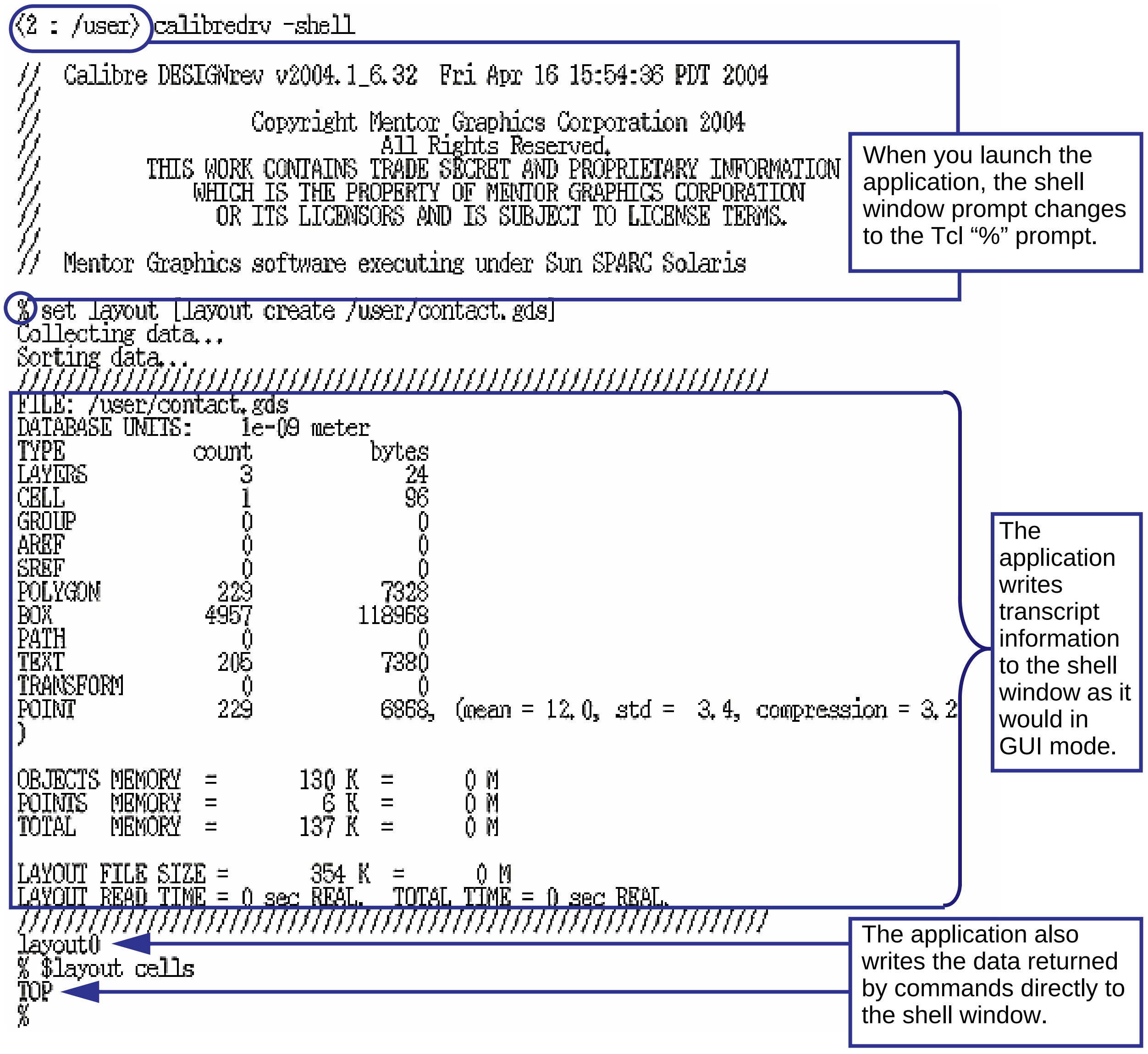
As specified in Table 3-1, you can run Calibre RVE from the command line with the -rve option. The layout viewer also invokes Calibre RVE when it first starts up and, if a valid Calibre RVE filename and argument(s) are specified, loads that file using the specified arguments. Note that the file and any arguments need to be interpreted as a single character string by Tcl, which requires you to enclose the string in quotes (““) or braces ([]). If you are supplying a filename with no arguments, the quote marks are optional.

Interactive Shell Mode

When you invoke in interactive shell mode, you have access only to the shell window. You issue commands by typing them directly into the window. While you can create and manipulate layout data, you cannot view it. Instead, you see descriptions of the data, reported in the shell window.

You invoke interactive shell mode by issuing the *basic* command[1] with the ***-shell*** argument.

Figure 3-2. Interactive Shell Mode



```
{2 : /user} calibredrv -shell
//  Calibre DESIGNrev v2004.1_6.32  Fri Apr 16 15:54:36 PDT 2004
//
//                 Copyright Mentor Graphics Corporation 2004
//                            All Rights Reserved.
//         THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
//            WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION
//              OR ITS LICENSORS AND IS SUBJECT TO LICENSE TERMS.
//
//  Mentor Graphics software executing under Sun SPARC Solaris
% set layout [layout create /user/contact.gds]
Collecting data...
Sorting data...
//////////////////////////////////////////////////////////////////
FILE: /user/contact.gds
DATABASE UNITS:    1e-09 meter
TYPE              count          bytes
LAYERS                3             24
CELL                  1             96
GROUP                 0              0
AREF                  0              0
SREF                  0              0
POLYGON             229           7328
BOX                4957         118968
PATH                  0              0
TEXT                205           7380
TRANSFORM             0              0
POINT               229           6868, (mean = 12.0, std =  3.4, compression = 3.2
)

OBJECTS MEMORY  =      130 K  =      0 M
POINTS  MEMORY  =        6 K  =      0 M
TOTAL   MEMORY  =      137 K  =      0 M

LAYOUT FILE SIZE =       354 K  =      0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.
//////////////////////////////////////////////////////////////////
layout0
% $layout cells
TOP
%
```

When you launch the application, the shell window prompt changes to the Tcl “%” prompt.
The application writes transcript information to the shell window as it would in GUI mode.
The application also writes the data returned by commands directly to the shell window.

1. calibrewb, calibredrv, calibrelv, or calibremdpv

Issuing Tcl Commands in Batch Mode

In batch mode, you have access to neither the shell nor the GUI. You can process either a single Tcl command or a full Tcl script. Once the application processes the command or script, it exits. You have no opportunities to issue other commands, or process additional scripts.

Your choices when running in batch mode are:

- Batch Mode — Tk is not available
- Batch GUI Mode — Tk is available

Batch Mode

In standard Batch mode, only the Tcl software package is available to you. You can use this mode any time you will not instruct the application to execute any Tk commands or any of the Calibre WORKbench simulation or modeling commands that require GUI mode.

```
calibredrv myscript.tcl
```

Batch GUI Mode

Batch GUI mode refers to batch processing in which both the Tcl and Tk software packages are available to you. You must use the batch GUI mode any time you issue either of the following:

- a Tk command.
- any of the Calibre WORKbench simulation or modeling commands that require GUI mode.

To run in batch GUI mode, **append** the command argument ***-gui*** to the end of the invocation command:

```
calibredrv myscript.tcl -gui
```

Table 3-2. Summary of Modes

Mode	**Displays GUI?**	**Uses Tk?**	**Invocation**	**Comments**
Interactive GUI	yes	yes	(no arguments)	Can see the data
Interactive	no	no	-shell	Can interact with the application
Batch	no	no	<script>	Can only use Tcl and non-GUI simulation/modelling commands
Batch GUI	no	yes	<script> -gui	Can use batch commands that require Tk.

Tcl and Shared Libraries

The Calibre tools are compatible with shared libraries (compiled for the correct host platform). You can use the Tcl `load` command as part of a Tcl script you run in Calibre WORKbench, Calibre LITHOview, DESIGNrev, and MDPview, specifying a path to the shared library.

Exercises in Modes of Operation

This section explores the differences you encounter when performing a single task in the different modes of operation.

- Exercise 1: Listing the Cells in a Layout in Interactive Shell Mode
- Exercise 2: Listing the Cells in an Open Layout in Interactive GUI Mode
- Exercise 3: Listing the Cells in a Layout Using Batch Mode

Exercise 1: Listing the Cells in a Layout in Interactive Shell Mode

Topics introduced:

- The Tcl `set` command to create and set a variable
- Command substitution
- Variable substitution (using variables in commands)
- The `layout create` object command
- The `$L cells` instance command

For this example, assume you do not have a layout loaded into the application. Assume that the file *sample.gds* exists in the directory from which you invoke the application.

1. Invoke Calibre DESIGNrev in shell mode:

   ```
   calibredrv -shell
   ```

2. When you see the “%” prompt, type the following command, then press **Enter**.

   ```
   set my_layout [layout create sample.gds]
   ```

   This line combines two commands:

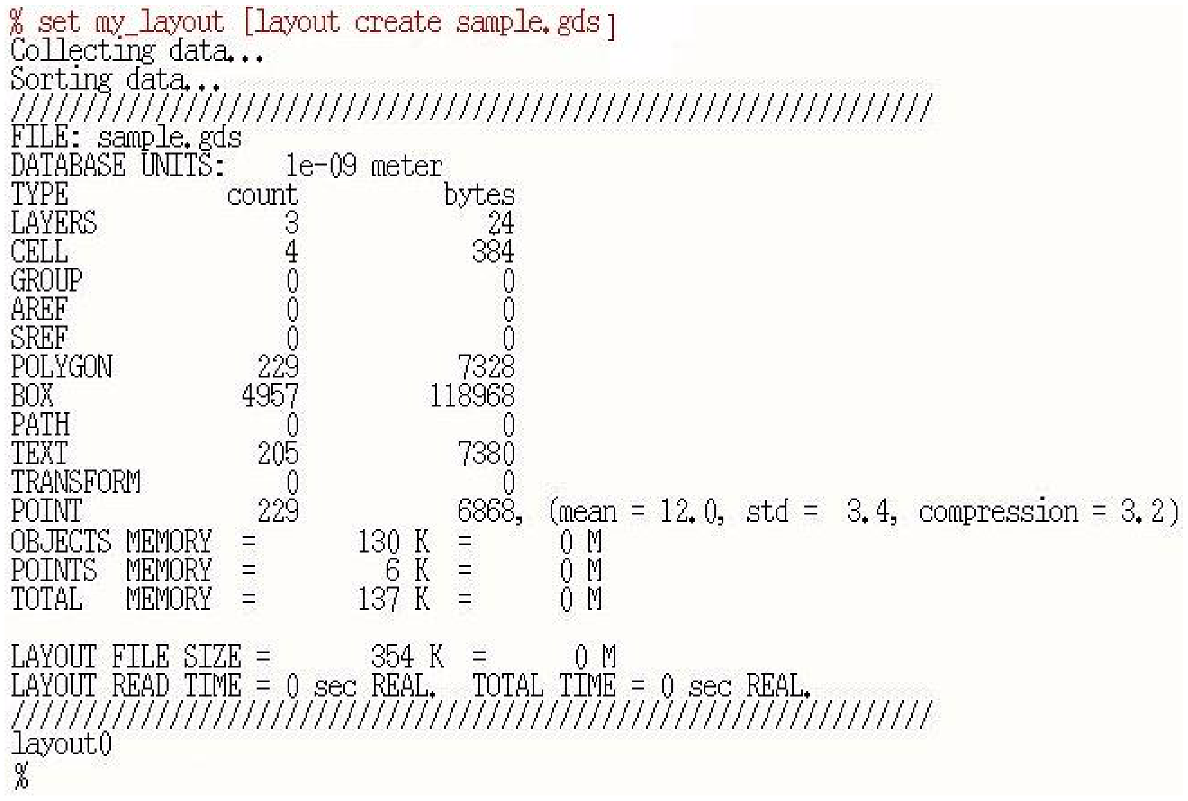
   - `set` — one of the Basic Tcl Commands. It creates a variable and assigns a value to it. In this case, the variable is called “my_layout”. The value is the data returned by the `layout create` command.
   - `layout create` — one of the Layout Object Class Commands. It reads the gds file into a layout object that it creates.

The *square brackets* instruct Tcl to replace the square brackets and everything between them with the results it gets by executing the command inside the brackets. This is called *command substitution*. In this case, the command it executes is `layout create`, and the result is "layout0", which is a system generated name (also called a *handle)* for the newly created layout. Thus "`set my_layout [layout create sample.gds]`" becomes "`set my_layout layout0`".

When you press **Enter** after typing the first line, Calibre DESIGNrev processes the command, displays the transcript information[1] generated when loading the layout, then prints the result of processing[2], in this case "layout0".

When processing is complete, it displays the "%", indicating it is ready for you to issue a new command.

Figure 3-3. Loading a Layout in Shell Mode



```
% set my_layout [layout create sample.gds]
Collecting data...
Sorting data...
////////////////////////////////////////////////////////////////
FILE: sample.gds
DATABASE UNITS:    1e-09 meter
TYPE            count        bytes
LAYERS              3           24
CELL                4          384
GROUP               0            0
AREF                0            0
SREF                0            0
POLYGON           229         7328
BOX              4957       118968
PATH                0            0
TEXT              205         7380
TRANSFORM           0            0
POINT             229         6868,  (mean = 12.0,  std =  3.4,  compression = 3.2)
OBJECTS MEMORY  =       130 K  =       0 M
POINTS  MEMORY  =         6 K  =       0 M
TOTAL   MEMORY  =       137 K  =       0 M

LAYOUT FILE SIZE =        354 K  =       0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.
////////////////////////////////////////////////////////////////
layout0
%
```

1. At the "%" prompt, type the following command, then press **Enter**.

   ```
   $my_layout cells
   ```

1. Whenever you run Calibre WORKbench, DESIGNrev, or MDPview, the application generates a transcript containing status information, errors, and warnings. For more information, refer to "Information in the Shell Window" in the appropriate User's Manual.
2. Printing the Tcl results to the terminal window is a performed by the tool in interactive mode only. In batch mode, the application only prints information to the terminal window if you explicitly program it to do so.

This line contains a single command. The first part of the command is `$my_layout`, which is the variable you created with a dollar sign ($) in front of it. The $ tells Tcl to substitute the value of the variable `my_layout` for the string "`$my_layout`". Thus, the line you typed translates into "`layout0 cells`".

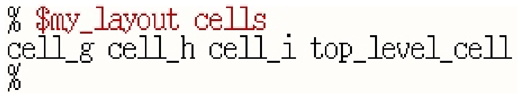
`$my_layout cells` is one of the Instance Commands. In Tcl, when you create an instance of an object you also enable a set of commands that operate on that instance (and only that instance). The name of the commands are always "<instance name> <operation>". Within the reference material for this document, instance command names are written using the format "$<variable name> <operation>", representing the command line you would use if you saved the instance name to the following variable names:

cwbWorkBench — cwb
Layout — L
cwbLayoutView — V
cwbGrid — grid
StringFeature — str

Thus, you would find reference information on `$my_layout cells` by looking up $L cells.

When the application evaluates the command "`layout0 cells`," it returns a list of the cells in layout0. In this case there are four: top_level_cell, cell_g, cell_h, and cell_i. When processing is complete, it displays the "%", indicating it is ready for you to issue a new command.

Figure 3-4. Listing Cells in Shell Mode



```
% $my_layout cells
cell_g cell_h cell_i top_level_cell
%
```

2. Exit the application.

Exercise 2: Listing the Cells in an Open Layout in Interactive GUI Mode

Topics introduced:

- iTcl `find objects` command
- Combining variable substitutions and command substitution

In this example, you work with a layout loaded into the Calibre DESIGNrev GUI. (Ignore the fact that the application displays the names of the cells in the layout for you. The lines of code you learn here will be needed elsewhere.)

In “Exercise 1: Listing the Cells in a Layout in Interactive Shell Mode”, you used the “`layout0 cells`” instance command to list the cells in layout0. Because the name of the command is of the form <instance name> <operation>, you cannot use this command unless you know the name of the layout instance. And since the layout instance name is system generated, the first thing you must do when listing cells in an open layout is to obtain the name the application assigned to the layout.

1. Invoke Calibre DESIGNrev in interactive GUI mode:

   ```
   calibredrv -m sample.gds
   ```

   The **-m** option instructs Calibre DESIGNrev to open the specified layout file in the Calibre database and the Calibre DESIGNrev GUI; in this case, this our sample file, “sample.gds”.

Note

In Exercise 1, you used the `layout create` command to create a layout in the Calibre database. However, `layout create` does not load the layout into the GUI (since it is not a Tk command). Use the $cwb viewedLayoutClbk command to load a layout file into the GUI.

2. Switch from the GUI window to the terminal window by clicking in the terminal window. You many need to move, resize, or minimize the GUI.
3. Press **Enter**.
4. When you see the “%” prompt, type the following command, then press **Enter**.

   ```
   set wb [find objects -class cwbWorkBench]
   ```

   This line combines two commands:

   - `set` — creates the variable `wb` and assigns it the result of the `find objects` command.

- `find objects` — finds any object of class cwbWorkBench. Since you have only one DESIGNrev open in this session, the command returns only one value, which is the system generated name for this object, in this case "`::cwbWorkBench::cwbWorkBench0`"[1].

Note

The examples in these sections assume that you only have one layout viewer open. If you have more than one window open, find objects returns multiple cwbWorkBench handles, one for each display window.

You need to know the name of the cwbWorkBench object because one of the functions it performs is to keep track of the names of layout objects.

When you press Enter after typing the first line, Calibre DESIGNrev prints the value of the new variable `wb`, in this case "`::cwbWorkBench::cwbWorkBench0`". It then displays the Tcl prompt, indicating that it is ready for the next command.

5. When you see the "%" prompt, type the following command, then press **Enter**.

```
set my_layout [$wb cget -_layout]
```

Here you use the variable `wb` as a shortcut to typing the name of the cwbWorkBench object, which is also the first part of the command you must use to obtain the layout name. The operation it performs is `cget` (get the value of a variable belonging to this class of object), and the argument `-_layout` is the name of the variable you want to get, which in this case is the variable that stores the handle for the current layout.

This command creates the variable `my_layout` and assigns it the result of the `$wb cget` command. It combines both forms of substitutions:

Variable substitution replaces `$wb` with the value of `wb`, resulting in

```
set my_layout [::cwbWorkBench::cwbWorkBench0 cget -_layout]
```

Command substitution replaces the square brackets and everything between them with the results of processing the command inside, resulting in:

```
set my_layout layout0
```

6. When you see the "%" prompt, type the following command, then press **Enter**.

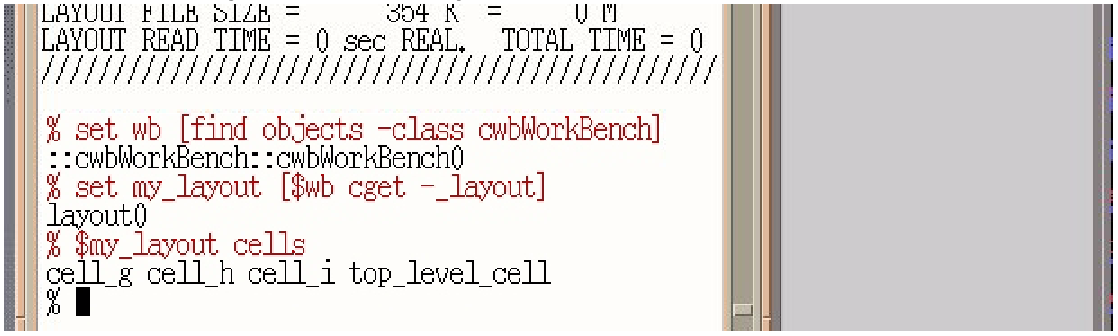
```
$my_layout cells
```

1. The double colon "::" is an itcl construct used to identify and manage namespaces. Unless you go on to advanced Tcl programming, you need not worry about itcl namespaces. Curious readers can find more information on namespaces at the SourceFORGE website: http://incrtcl.sourceforge.net/itcl/namesp.html.

Because of variable substituting, the line you typed translates into the instance command "`layout0 cells`".

When Calibre DESIGNrev evaluates the command "`layout0 cells`", it returns a list of the cells in layout0, then displays the "%", indicating it is ready for you to issue a new command.

Figure 3-5. Listing Cells in Interactive GUI Mode



```
LAYOUT FILE SIZE =        354 K   =      0 M
LAYOUT READ TIME = 0 sec REAL.   TOTAL TIME = 0
/////////////////////////////////////////////////

% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set my_layout [$wb cget -_layout]
layout0
% $my_layout cells
cell_g cell_h cell_i top_level_cell
%
```

Note

In both Exercise 1: Listing the Cells in a Layout in Interactive Shell Mode and Exercise 2: Listing the Cells in an Open Layout in Interactive GUI Mode, it would have been just as easy to skip the variables and simply retype the values returned by various Tcl commands. However, in most real-life situations, you will be doing much more complex work, in which using variables becomes necessary. When writing Tcl scripts, variables are a necessity.

7. Exit the application.

Exercise 3: Listing the Cells in a Layout Using Batch Mode

Topics introduced:

- Displaying results using the **puts** basic Tcl command

For this example, instead of typing the Tcl commands at a prompt, you write them as an ASCII file, save the file, then pass the file to Calibre DESIGNrev.

If you wanted to list the cells in a layout using batch mode, the logical thing to do would seem to be to write a script that contains only those commands that you issued at the prompt when working in interactive shell mode.

Unfortunately, there is a problem with this approach. Tcl discovers the names of the cells that you are looking for, but it does not display them for you to read! When working in interactive shell mode, the shell does the displaying for you, and prints the return values for every command you issue. In batch mode, Tcl only writes information to the terminal window when you explicitly tell it to do so.

1. Edit the Tcl script, adding the **puts** command to the last line, and save as a new file, *DisplayCellList.tcl*.

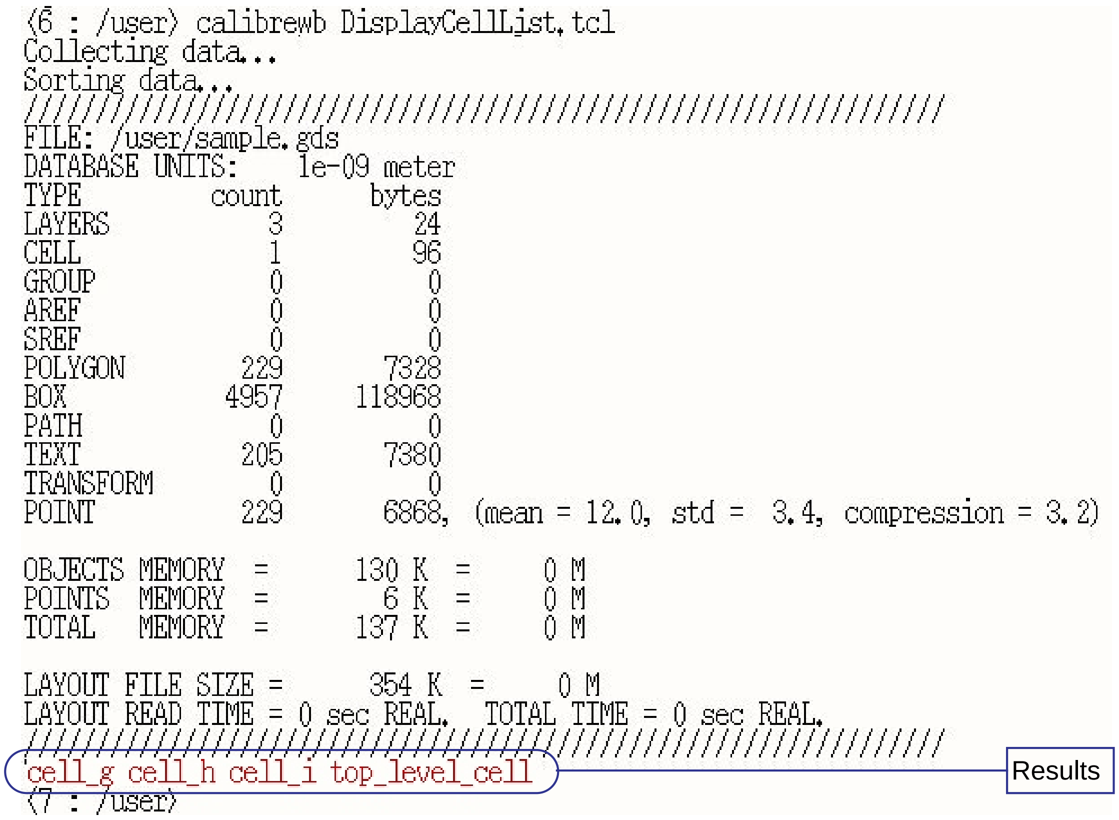
   ```
   set my_layout [layout create sample.gds]
   puts stdout [$my_layout cells]
   ```

   The basic Tcl command **puts** takes two arguments, the output channel (where to print the information) and the data (what to print). Because the purpose here is to display the cells list, the output channel is `stdout`, which defaults to the terminal window.

2. Invoke Calibre DESIGNrev in batch mode by passing in the name of the script:

   ```
   calibredrv DisplayCellList.tcl
   ```

Figure 3-6. Second Attempt at Listing Cells in Batch Mode



```
<6 : /user> calibrewb DisplayCellList.tcl
Collecting data...
Sorting data...
////////////////////////////////////////////////////////////////
FILE: /user/sample.gds
DATABASE UNITS:    1e-09 meter
TYPE          count      bytes
LAYERS            3         24
CELL              1         96
GROUP             0          0
AREF              0          0
SREF              0          0
POLYGON         229       7328
BOX            4957     118968
PATH              0          0
TEXT            205       7380
TRANSFORM         0          0
POINT           229       6868,  (mean = 12.0, std =  3.4,  compression = 3.2)

OBJECTS MEMORY  =      130 K  =     0 M
POINTS  MEMORY  =        6 K  =     0 M
TOTAL   MEMORY  =      137 K  =     0 M

LAYOUT FILE SIZE =      354 K  =     0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.
////////////////////////////////////////////////////////////////
cell_g cell_h cell_i top_level_cell
<7 : /user>
```

Examples In Writing Procedures

This section of the tutorial explores the basics of writing procedures, using examples that display an entire layout hierarchy. For this set of examples, assume that the file *std_des.gds* exists in the directory from which you invoke the application, and contains three levels of hierarchy.

- Example 1: Using a Simple Procedure to Write the Layout Hierarchy
- Example 2: Using a Recursive Procedure to Write a Layout Hierarchy

Example 1: Using a Simple Procedure to Write the Layout Hierarchy

Topics introduced:

- Writing results to a file
- Procedures
- Using the `foreach` Tcl command to look through an array of objects
- Displaying the contents of a file inside the DESIGNrev terminal window

For this example, assume you have a *std_des.gds* loaded into Calibre DESIGNrev.

1. Invoke Calibre DESIGNrev in GUI mode, loading in *std_des.gds*.

   ```
   calibredrv -m std_des.gds
   ```

2. Switch from the GUI window to the terminal window.

3. Press **Enter** to access the Tcl "%" prompt.

4. Enter the following lines to open a file to which you will write the hierarchy data:

   ```
   set fileID [open dump_hier.txt w]
   puts $fileID “This file displays the hierarchy for std_des.gds”
   ```

   In Interactive mode, DESIGNrev writes the results to the terminal window. However, because listing the hierarchy involves several commands, the results are interspersed between command lines. Writing to a file allows you to view the complete hierarchy without any extraneous information.

   The basic Tcl command `open` opens a file and returns the *channelID* it must use to communicate with that file. The argument `w` (write) tells Tcl to open the file as write-only; Tcl will create the file if it does not exist.

   The `puts` command does the actual writing to the file. Remember that this command takes two arguments, the output channel (where to print the information) and the data (what to print). Using the *channelID* as the first argument instructs the command to write the data to the file rather than the terminal window (stdout). The string enclosed in quotes is the data to write to the file. Since the whole string is the second argument, you must enclose it in quotes, telling the command to treat all the words in the string as a single object.

5. Get the layout name:

```
set wb [find objects -class cwbWorkBench]
set layout [$wb cget -_layout]
```

6. Write the topcell of the layout to the file. This is the first level of hierarchy.

```
puts $fileID “Topcell: [set top [$layout topcell]]”
```

As in the previous `puts` command, the string enclosed in quotes is the data written to the file. However, this string is different. Even though there are quotes around the string, Tcl recognizes the square brackets as indicating the need for command substitution and the $ as indicating need for variable sustentation. This `puts` command writes the string “Topcell: “ followed by the value or values returned by processing the `set` command, which is the value it assigns to the variable.

Since the `set` command is also written using command substitution, the variable `top` gets set to the results of processing the instance command `$layout topcell`. This command returns only the name of the topcell, rather than the full list of cells you listed in the previous exercises.

Note that if you wanted to print special characters such as “[“ “]” or “$”, you could do so by preceding them with a backslash “\”.

7. Next, write a procedure to print out the children of a given cell. You will invoke the procedure later. Here you define the procedure by specifying what it is called, the arguments it takes, and what it does.

```
proc list_children { L F C} {
     puts $F “Children of $C: [$L children $C]”
}
```

Procedures always take the form:

proc <procedure name> <arguments list> <body>

- proc is a basic Tcl command that creates a procedure.
- The **name** of this procedure is “list_children”.
- The **arguments list** is always enclosed in braces ({ }). This argument list contains three arguments: L, F, and C. Within the body of the procedure, these arguments will be used as variables.
- The **body** is also enclosed in braces. The Tcl interpreter understands that everything between the open and close braces forms the body of the procedure,

even if it spans multiple lines and contains multiple commands. Think of this as a mini-script. It contains any commands needed to perform the processing.

The body of this procedure prints the string “Children of” followed by the name of the cell you are processing, followed by the list of children.

8. Invoke the procedure to write the children of the topcell to the file. The children of the topcell form the second level of hierarchy.

```
list_children $layout $fileID $top
```

9. Invoke the procedure again to write any children of the children of topcell to the file. The children of the children form the third level of the hierarchy. To avoid issuing the procedure for every child cell, use the **foreach** command to cycle through them.

```
foreach cell [$layout children $top] {
     list_children $layout $fileID $cell
}
```

The **foreach** command creates a *loop variable* (cell) and assigns it the values resulting from processing the command in square brackets. It cycles through the values one at a time, evaluating the commands between the curly braces.

10. Because you are working in interactive GUI mode, you can look at the cell hierarchy list in the GUI to see if there are more levels of the hierarchy. In this case, you see that *std_des.gds* has only three levels of hierarchy. If the layout contained four levels of hierarchy, you would add another **foreach** calling the procedure to list that next level:

```
foreach cell [$layout children $top] {
   foreach child [$layout children $cell] {
        list_children $layout $fileID $child
   }
}
```

Since the third level of the hierarchy contains two cells, NAND2A and NOR, this is equivalent to the following two **foreach** statements:

```
foreach cell [$layout children NAND2A] {
   list_children $layout $fileID $cell
}
foreach cell [$layout children NOR] {
   list_children $layout $fileID $cell
}
```

Working in interactive GUI mode, you can continue to add these looping statements as needed to cover the walk through the entire hierarchy. Note that this is an inefficient way to solve this problem, because it depends on you being able to anticipate how many levels of hierarchy the layout contains. For a more efficient approach, refer to “Example 2: Using a Recursive Procedure to Write a Layout Hierarchy”.

11. Save the file:

    ```
    close $fileID
    ```

12. Reopen the file so you can view its contents:

    ```
    set fileID [open dump_hier.txt r]
    read $fileID
    ```

    Since you had to close the file to save it, you must open it again. This time you use the argument `r` (read), which tells Tcl you want to read the data in the file.

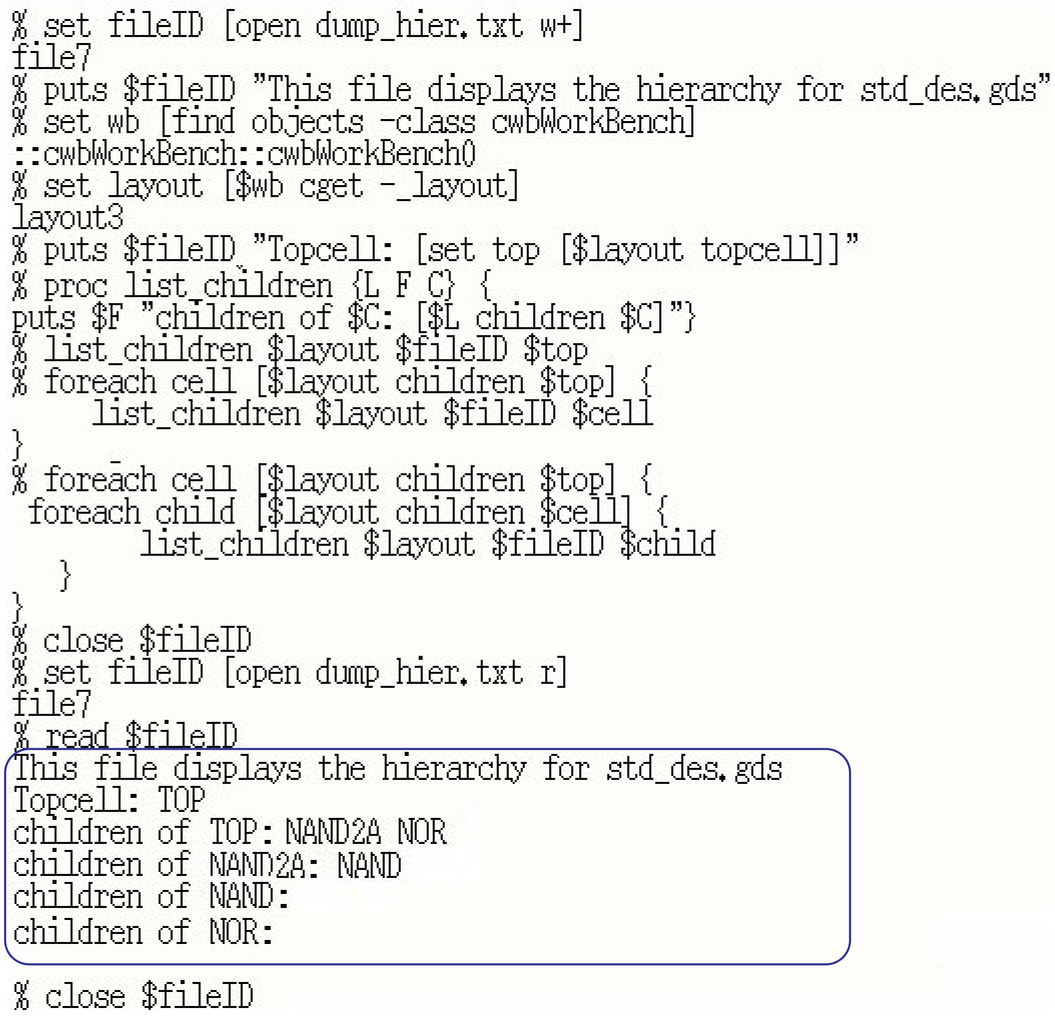
    The basic Tcl command `read` requires at least one argument, the input channel (where to find the information.) It returns the information it reads to the application. In either interactive mode, the application prints the return values for all commands to the shell, displaying it for you. If you were writing a script to be executed in batch mode, you would need to write the `read` command this way:

    ```
    set data [read $fileID]
    puts $data
    ```

13. Close the file:

    ```
    close $fileID
    ```

    It is good practice to close any file you open.



```
% set fileID [open dump_hier.txt w+]
file7
% puts $fileID "This file displays the hierarchy for std_des.gds"
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set layout [$wb cget -_layout]
layout3
% puts $fileID "Topcell: [set top [$layout topcell]]"
% proc list_children {L F C} {
puts $F "children of $C: [$L children $C]"}
% list_children $layout $fileID $top
% foreach cell [$layout children $top] {
     list_children $layout $fileID $cell
}
% foreach cell [$layout children $top] {
 foreach child [$layout children $cell] {
        list_children $layout $fileID $child
   }
}
% close $fileID
% set fileID [open dump_hier.txt r]
file7
% read $fileID
This file displays the hierarchy for std_des.gds
Topcell: TOP
children of TOP: NAND2A NOR
children of NAND2A: NAND
children of NAND:
children of NOR:

% close $fileID
```

Example 2: Using a Recursive Procedure to Write a Layout Hierarchy

Topics introduced:

- Comments
- Procedure arguments with default values
- Recursive Procedures
- Local vs. Global variables

For this example, you will write a script that writes the layout hierarchy for any layout file, regardless of how many levels there are in the hierarchy. You do this by writing a procedure that traverses the hierarchy until it can go no further.

Explanations follow the example code.

1. Copy the following script into a text file named *hier_dump.tcl*.

```
##############################################################
# PROC: dump_hier
# L      = layout object name
# F      = file to dump hierarchy to
# C      = cell to start dumping from, "" means use topcell
# Indent = string used to format the hierarchy
##############################################################

proc dump_hier { L  F {C ""} {Indent ""} } {
     if {$C == ""} {
       set C [$L topcell]
     }
     puts $F “$Indent --> $C”
     append Indent “        “
    foreach child [$L children $C] {
                         dump_hier $L $F $child $Indent
     }
}

set fileID [open dump_hier.txt w]

#
# Step 1: open the layout
#

set layout [layout create std_.gds]

#
# Step 2: write the hierarchy to the file
#

dump_hier $layout $fileID

#
# Step 3: Close (and save the file)
```

```
#

close $fileID
```

2. Invoke DESIGNrev:

```
calibredrv hier_dump.tcl
```

3. View the file you have created:

```
more dump_hier.txt
```

Comments as Program Information

Comments in a Tcl program begin with the pound sign “#”. Any time you write a Tcl script, you should use comments to explain what you are doing and tell a future user how to use the script.

These first seven lines of the script are comment lines that introduce the procedure that follows, defining the name of the procedure and the arguments it takes.

```
###############################################################
# PROC: dump_hier
# L      = layout object name
# F      = file to dump hierarchy to
# C      = cell to start dumping from, "" means use topcell
# Indent = string used to format the hierarchy
###############################################################
```

Procedures

The next nine lines contain the procedure itself.

```
proc dump_hier { L  F {C ""} {Indent ""} } {
     if {$C == ""} {
        set C [$L topcell]
     }
     puts $F “$Indent --> $C”
     append Indent “        “
     foreach child [$L children $C] {
                        dump_hier $L $F $child $Indent
     }
}
```

Notice that this list of arguments is different than the list of arguments in the procedure you wrote in the previous exercise. It contains four arguments:

{ L F {C ""} {Indent ""} }

The first two arguments, L and F, are represented by their names alone because they have no default values. The second two arguments, C and Indent, are enclosed in braces because they have default values, in both cases an empty string.

The body is also more complex. First, it checks to see if you passed it a cell name. If you did not pass it a cell name, the value of C will be " ", which is the default value. In this case, the procedure extracts the cell name using the $layout topcell instance command.

```
if {$C == ""} {
   set C [$L topcell]
}
```

Next the procedure writes the name of the cell it is processing to the output file. $Indent and "-->" add formatting to the output to help visualize the layout hierarchy.

```
puts $F "$Indent --> $C"
```

The Tcl command **append** adds more characters to an existing variable (in this case, $Indent). Here it is used to show the layout hierarchy graphically, forcing the name of a child cell to be indented relative to the name of the parent cell.

```
append Indent "        "
```

The last thing this procedure does is cycle through the children of the cell, invoking itself to print their names and children.

```
foreach child [$L children $C] {
                   dump_hier $L $F $child $Indent
}
```

When a procedure calls itself, it is called a *recursive procedure*. It will call itself as many times as needed, until it reaches a cell that has no children. It then pops back up one level of the hierarchy and looks for children of the next cell.

Variables

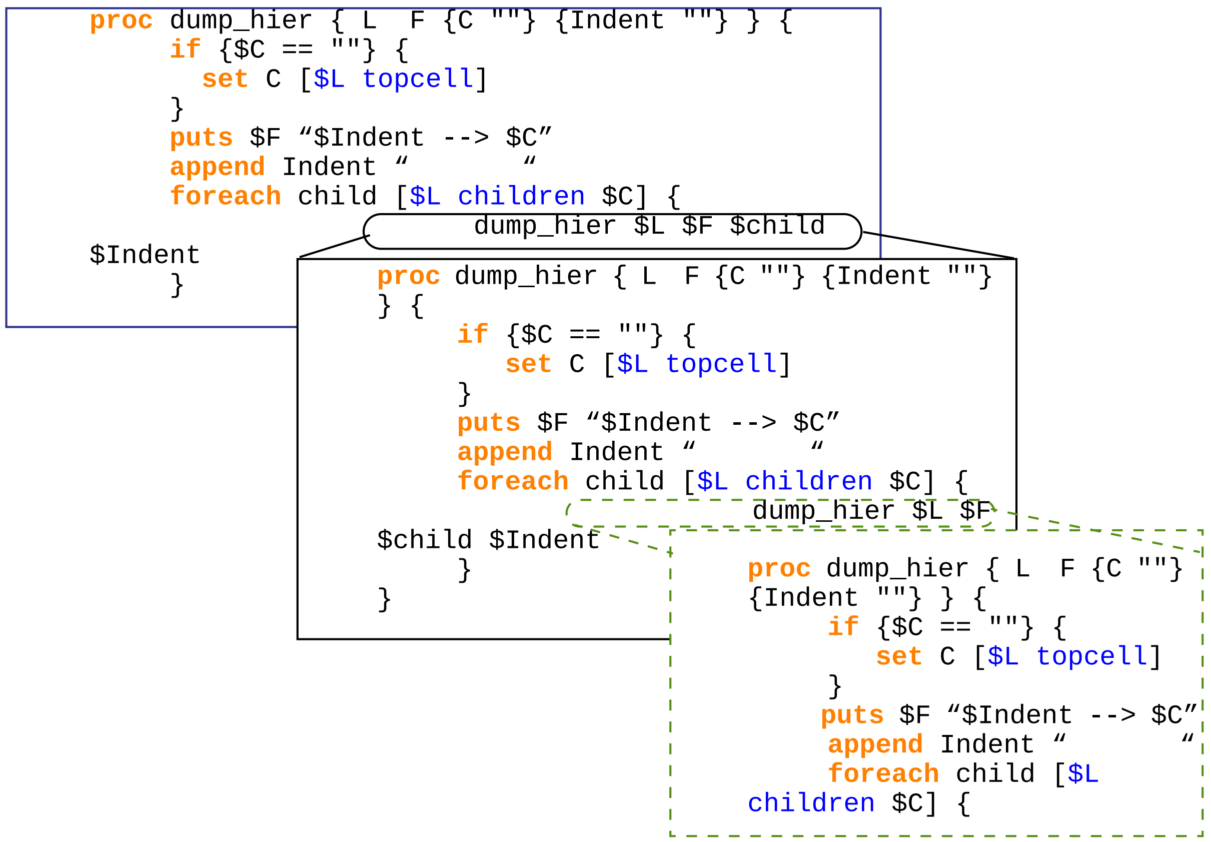
Recursive procedures are possible because Tcl supports the concept of variable scope. Scope defines where a variable has meaning. Scope can be either local or global:

- *Local variables* are variables that exist (have meaning) only within a single procedure. When you execute a procedure, Tcl creates a local variable for each of the arguments to that procedure. While Tcl is processing the body of the dump_hier procedure, it knows what $L is. Once it exits the procedure, $L has no meaning.
- *Global variables* are variables that have meaning anywhere with the program[1].

When the recursive procedure calls itself, it launches a separate procedure, which happens to be another instance of the same procedure. The result is procedure within a procedure, within a procedure.

1. To use a global variable within a procedure, you must use special Tcl commands, such as `global` and `upvar`.

Figure 3-7. Graphical Representation of a Recursive Procedure



```
    proc dump_hier { L  F {C ""} {Indent ""} } {
         if {$C == ""} {
            set C [$L topcell]
         }
         puts $F "$Indent --> $C"
         append Indent "        "
         foreach child [$L children $C] {
                    dump_hier $L $F $child
$Indent
         }
```

```
     proc dump_hier { L  F {C ""} {Indent ""}
     } {
          if {$C == ""} {
             set C [$L topcell]
          }
          puts $F "$Indent --> $C"
          append Indent "        "
          foreach child [$L children $C] {
                           dump_hier $L $F
     $child $Indent
          }
     }
```

```
     proc dump_hier { L  F {C ""}
     {Indent ""} } {
          if {$C == ""} {
             set C [$L topcell]
          }
          puts $F "$Indent --> $C"
          append Indent "        "
          foreach child [$L
     children $C] {
```

All the variables in your recursive procedure are local. That means the variable F in the lowest level procedure, shown outlined with a green dotted line in Figure 3-7, is a completely different variable from the variable F in the middle procedure, shown outlined with a red solid line in Figure 3-7.

Questions and Answers

Q: I'm running Calibre WORKbench, not DESIGNrev. All the exercises tell me to invoke DESIGNrev. Where will I find exercises that apply to me?

A: All these exercises do apply to you. DESIGNrev provides a subset of the capabilities provided by the WORKbench tool. That means any scripts that run in Calibre DESIGNrev will run in Calibre WORKbench. It also means you can invoke either program to work through the exercises.

Q: This manual says that when I run Calibre DESIGNrev, I create a WORKbench object. Shouldn't I be creating a DESIGNrev object?

A: There is no DESIGNrev object. The basic application window object is called cwbWorkBench, whether you are running Calibre DESIGNrev, WORKbench, MDPview, or

LITHOview. The cwbWorkBench object is configured differently to create the different applications. Note also, that some of the hardcoded variable names and handles begin with “cwb”. These also apply to all the applications, not just WORKbench.

Q: I did some research on using Tcl, and read that I must always start a Tcl script with #! followed by the path to the Tcl interpreter. Why don’t the scripts in this manual start with #! ?

A: The scripts you pass to Calibre WORKbench for processing are not stand-alone scripts. They are actually chunks of code that get evaluated by the WORKbench program, which did begin with #!. This means that if you try to run your scripts outside of WORKbench, they will not work.

Q: What is the “\” I see at the end of some lines?

A: The backslash “\” at the end of a line indicates that the command would not fit onto one line, and that the line the follows finishes the command. When you use the backslash as a command continuation character, it must not be followed by any other characters. Even a space after it will create an error.

Chapter 4
Writing WorkBench Scripts

Tcl scripts are stand-alone programs that you write using the Tcl/Tk language, plus any extensions that you have available to you. *WorkBench*[1] *scripts* are Tcl scripts written using the Calibre WORKbench Tcl extensions and batch commands.

You can run WorkBench scripts either of two ways:

- In batch mode, by including the pathname for the script in the WORKbench invocation:

```
calibredrv my_script.tcl
calibredrv my_script.tcl -gui
```

- In interactive mode, by sourcing the script from the command line:

```
% source my_script.tcl
```

Note

Writing scripts for Calibre WORKbench, DESIGNrev, and MDPview involves working with the Mentor Graphics extensions to Tcl/Tk. The proprietary programming objects you are most likely to use in scripts are:

cwbWorkBench Object — the main layout editor widget
Layout Object — the Tcl object containing the layout data

1. Here, WorkBench is used as a generic term for the set of applications created with the cwbWorkBench class of objects. These are: Calibre WORKbench, DESIGNrev, LITHOview, and MDPview.

Scripts for Layout Generation and Assembly

Layout assembly takes existing cells or layout files and combines them.

Scripted Layout Generation involves creating or modifying a layout using batch processing. Scripted layout generation has many applications, including:

- Automatic layout generation.
- Automated conversion of data with scaling operations and/or changing database units.

Scripted Layout Assembly involves taking existing cells or layout files and combining them into a new layout. Assembly is a type of very high level layout editing – it never gets down to the polygon level. Assembly could be used in a variety of situations, such as:

- Combining several layouts into a test chip.
- Adding scribe-line data to the chip data for a mask.

Script 1: Writing a Script to Generate a New Layout

Topics introduced:

- The layout object as an empty container
- Creating cells, layers, and polygons in a layout
- Saving a layout as a GDS file

1. Create a Tcl script, called *new_layout.tcl* and add the following line of code:

   ```
   set L [layout create]
   ```

   This line combines two commands:

   - `set` — creates the variable `L` and assigns it the return value from the layout create command.
   - `layout create` — Creates the new layout. This command returns the handle of the newly created layout.

   This newly created layout is an empty container. It contains nothing; no cells and no layers. In the rest of this exercise, you will add data to this new layout.

2. Create a top cell within this layout:

   ```
   $L create cell TOP
   ```

   The `$L create cell` command creates the cell TOP in the new layout. It will be a top cell as long as it is not referenced within any other cells.

3. Create a few layers in the layout:

```
$L create layer 1
$L create layer 2.7
```

The `$L create layer` command lets you create layers using either an integer layer number or using the <layer>.<datatype> syntax, which defines a specific data type on a layer.

4. Add two polygons to layer 1:

```
$L create polygon TOP 1 0 0 100 100
$L create polygon TOP 1 0 0 0 40 30 40 30 80 45 80 45 25 15 25 15 0
```

The first `$L create polygon` command creates a rectangle, with opposite corners at (0,0) (100,100). The second creates a polygon with eight vertices.

Save the new layout to a GDS file:

```
$L gdsout temp.gds
```

5. Save the script.

6. Run the script:

```
calibredrv new_layout.tcl
```

7. View the new layout in Calibre DESIGNrev:

```
calibredrv -m temp.gds
```

Script 2: Setting up a Script to Accept Passed In Arguments

Topics introduced:

- `argv`
- `lindex`
- Using the return value from set
- `argc`
- `lrange`
- Passing arguments to a Tcl script

For this example, you will expand upon the previous script, *new_layout.tcl*. Assume that you want to pass arguments to the script that define the name of the top cell, a layer number, the coordinates of a polygon, and the name to use for saving the new gds layout.

1. Open *new_layout.tcl* and edit the second line:

   change: `$L create cell TOP`

   to:
   ```
   set my_cell [lindex $argv 0]
   $L create cell $my_cell
   ```

   The Tcl command `lindex` returns one or more elements in a Tcl list. In this case, the list is `argv`, which is a list managed by the Tcl interpreter. It contains all the arguments that were passed to a Tcl script.

   In Tcl, all variables are strings. The Tcl interpreter recognizes how to handle the contents of the variables based on the commands you use to manipulate them. For example:

   - `expr` — treat the string as a number
   - `lindex` — treat the string as a list, with spaces separating the items in the list
   - `puts` — treat the string as a string

Tip: In Tcl, the first element in a list has an index of 0. The last element in a list of length N has an index of (N-1).

   Thus, the command `set my_cell [lindex $argv 0]` extracts the first argument passed to the script for use as the cell name in the next line of code.

2. Edit the lines that created the layers to (a) create only one layer and (b) get the layer number from the arguments list:

   change:
   ```
   $L create layer 1
   $L create layer 2.7
   ```

to: `$L create layer [set layer_num [lindex $argv 1]]`

This line contains three nested commands:

- `lindex $argv 1` — return the second argument in the arguments list.
- `set layer_num [lindex $argv 1]` — process the command enclosed in square brackets, and store the results in the variable layer_num.
- `$L create layer [set layer_num [lindex $argv 1]]` — create the layer using the results from processing the command enclosed in square brackets as argument to the command. Since the return value for the `set` command is always the value to which the variable is set, the command uses the second argument passed to the script as the layer number.

3. Add the following code directly after the command to create the layer. This code extracts the polygon coordinates from the arguments list:

```
set first_coord 2
set last_coord [expr $argc-2]
set coords [lrange $argv $first_coord $last_coord]
```

To extract the coordinates of the polygon from the list, you use the `lrange` command. This command returns a new list containing all the elements with indices from `$first_coord` through `$last_coord`.

If all the arguments between the layer number and the name of the file are to be used as the coordinates of the polygon, you know that `first_coord`, the index of the first coordinate, must be 2 (since 0 was the topcell and 1 was the layer number).

The variable `argc` is a special variable managed by the Tcl interpreter that contains a count of the number of arguments passed to the script. You may recall that the index of the last element in a list of length N is (N-1). Since the last argument to be passed in will be the name of the file, the index of the final coordinate of the polygon must be (`$argc` -2).

The `expr` command instructs the Tcl interpreter to evaluate a mathematical operation and in doing so, treats the values stored in the specified variables as numbers rather than as strings. Thus, you can calculate `last_coord` using `[expr $argc-2]`.

4. Delete the line that creates the rectangle, and edit the line that creates the polygon:

change: `$L create polygon TOP 1 0 0 100 100`
`$L create polygon TOP 1 0 0 0 40 30 40 30 80 45 80 45 25 15 25 15 0`

to: `eval $L create polygon $my_cell $layer_num $coords`

The `eval` command is a special command the builds a command string out of a set of arguments and passes it to the Tcl interpreter for processing. When you must build a command string using variable substitution and/or command substitution, you sometimes end up with data in a format that does not match the command. For instance, in this case, `$L create polygon` expects each coordinate value to be a separate string, but you are passing in a single string(`$coords`) containing all of them. The `eval` command expands all the variables when it builds the command string, so the Tcl interpreter is able to process the `$L create polygon` command.

5. Edit the final line, saving the file to the file you pass in as the last argument:

   ```
   $L gdsout [lindex $argv end]
   ```

   The command `lindex` recognizes the string `end` as representing the last index in a list.

6. Save the script.

7. Run the script:

   ```
   calibredrv new_layout.tcl topcell 2 0 0 0 40 30 40 30 80 45 80 45 25
   15 25 new_file.gds
   ```

8. View the new layout in Calibre DESIGNrev:

   calibredrv -m new_layout.tcl

Completed Script new_layout.tcl

```
set L [layout create]
set my_cell [lindex $argv 0]
$L create cell $my_cell
$L create layer [set layer1 [lindex $argv 1]]
set first_coord 2
set last_coord [expr $argc - 2]
set coords [lrange $argv $first_coord $last_coord]
eval $L create polygon $my_cell $layer1 $coords
$L gdsout [lindex $argv end]
```

Script 3: Debugging and Adding Error Handling

Topics introduced:

- Deciphering Tcl error messages
- **llength**
- Numeric expressions as conditions for `if` statements
- The remainder operation “%”
- Basic error handling

For this example, you will introduce errors in *new_layout.tcl*, then find and fix them for a more realistic programming experience. You will also anticipate possible user errors and build in error handling for those situations.

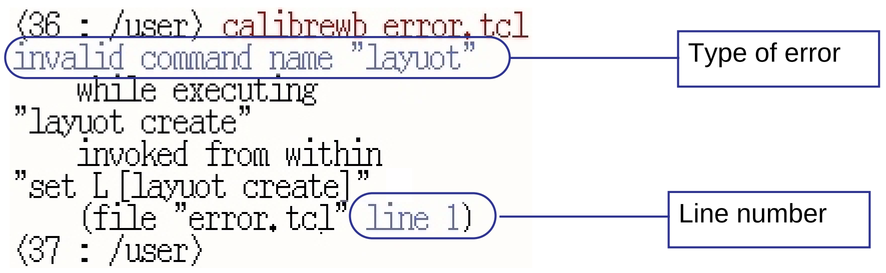
1. Modify *new_layout.tcl*, as shown in red, and save the script as *error.tcl*:

```
set L [layuot create]
set my_cell [lindex $argv 0]
$L1 create cell $my_cell
$L create layer [set layer1 [lindex $argv 1]]
set first_coord 2
set last_coord [expr $argc - 2]
set coords [lrange $argv $first_coord $last_coord]
$L create polygon $my_cell $layer1 $coords
$L gdsout [lindex $argv end]
```

2. Run the script.

```
calibredrv error.tcl
```

The Tcl interpreter encounters the first error and stops. It writes an error message to the shell window.

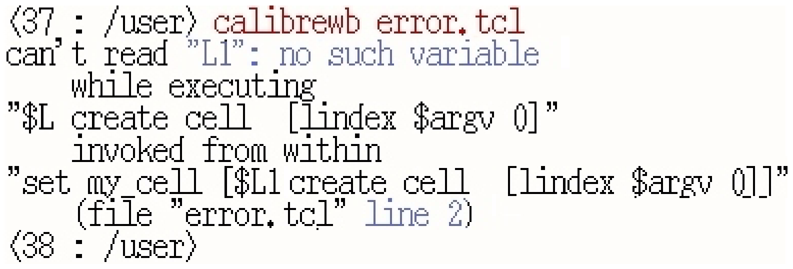


```
(36 : /user) calibrewb error.tcl
invalid command name "layuot"
    while executing
"layuot create"
    invoked from within
"set L [layuot create]"
    (file "error.tcl" line 1)
(37 : /user)
```

The Tcl interpreter recognizes “layuot” as a command name because it is at the beginning of the line.

3. Edit *error.tcl* to correct the spelling mistake in the first line, save the file, then run the script again.

```
calibredrv error.tcl
```

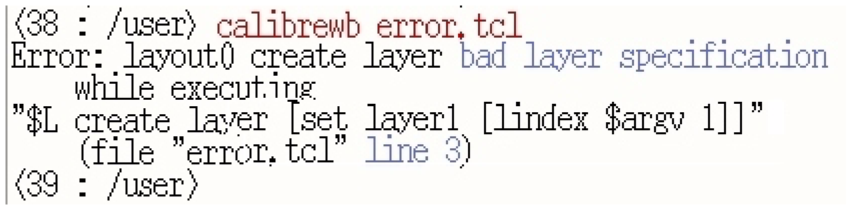


```
<37 : /user> calibrewb error.tcl
can't read "L1": no such variable
    while executing
"$L create cell  [lindex $argv 0]"
    invoked from within
"set my_cell [$L1 create cell  [lindex $argv 0]]"
    (file "error.tcl" line 2)
<38 : /user>
```

The Tcl interpreter recognizes that the error is in a variable name, because it encountered the $.

4. Edit *error.tcl*, changing $L1 to $L in the second line, save the file, then run the script again.

```
calibredrv error.tcl
```



```
<38 : /user> calibrewb error.tcl
Error: layout0 create layer bad layer specification
    while executing
"$L create layer [set layer1 [lindex $argv 1]]"
    (file "error.tcl" line 3)
<39 : /user>
```

The error message this time suggests that the layer specification is bad. But you did not edit this line, and the code worked fine before.

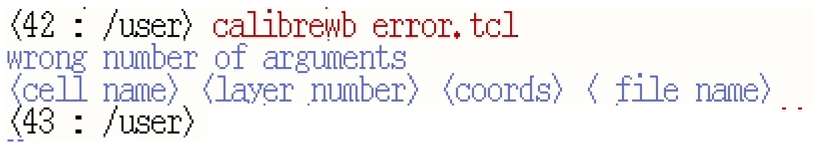
The problem here is that `[set layer [lindex $argv 1]]` has no meaning if `argv` is empty, which it is, because you did not pass in any arguments.

5. Add the following code to the beginning of the script:

```
############################################################
#
# This script requires the following arguments:
#  arg 0 --> cell name
#  arg 1 --> layer number
#  args 2 through n --> coordinates for polygon
#                         (minimum of 4 values)
#  arg (n+1) --> file name
#
############################################################

if {$argc < 7} {
    puts "wrong number of arguments"
    puts "<cell name> <layer number> <coords> < file name>"
    exit
}
```

This code checks to see that you passed in enough arguments. If not, it tells the user exactly what arguments to supply and exits the program.

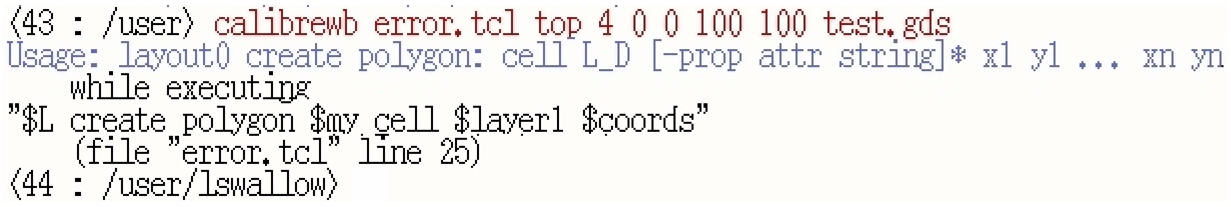
6. Save the file, then run the script again with no arguments.



```
<42 : /user> calibrewb error.tcl
wrong number of arguments
<cell name> <layer number> <coords> < file name>
<43 : /user>
```

From now on, you must supply at least seven arguments to run or debug your script.

7. Run the script again, this time supplying the minimum required arguments:



```
<43 : /user> calibrewb error.tcl top 4 0 0 100 100 test.gds
Usage: layout0 create polygon: cell L_D [-prop attr string]* x1 y1 ... xn yn
    while executing
"$L create polygon $my_cell $layer1 $coords"
    (file "error.tcl" line 25)
<44 : /user/lswallow>
```

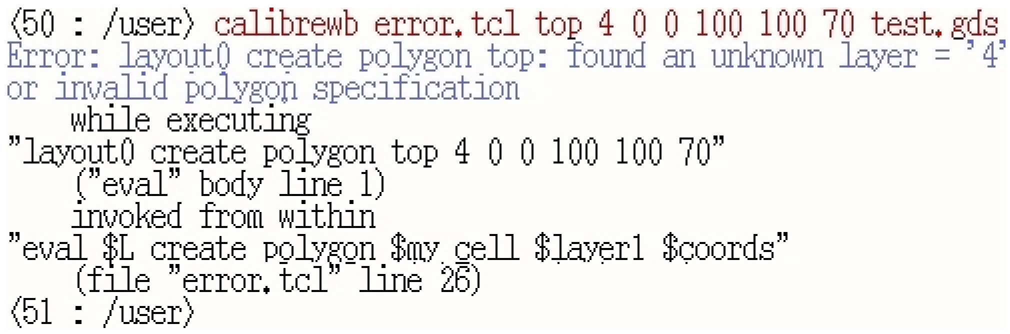
This error indicates that the Tcl interpreter cannot decipher the `$L create polygon` command. Notice that it does not give you a lot of information about why it could not execute the command. When a command that uses variable and/or command substitution does not execute, try using **`eval`** before you assume that the data is wrong.

If the **`eval`** command did not solve the problem, you would look at each of the variables, in this case `$my_cell`, `$layer`, and `$coords`, and make sure they are passing in the correct information.

8. Add the **`eval`** command back in front of `$L create polygon`, save the script and run it again.

   The script should run without errors.

9. Run the script again, this time passing in an odd number of coordinate values.



```
<50 : /user> calibrewb error.tcl top 4 0 0 100 100 70 test.gds
Error: layout0 create polygon top: found an unknown layer = '4'
or invalid polygon specification
    while executing
"layout0 create polygon top 4 0 0 100 100 70"
    ("eval" body line 1)
    invoked from within
"eval $L create polygon $my_cell $layer1 $coords"
    (file "error.tcl" line 26)
<51 : /user>
```

An odd number of coordinate values cannot define a polygon, hence you get an error message when the Tcl interpreter attempts to evaluate the `$L create polygon` command. To avoid this problem, you need to check the number of coordinates before proceeding.

10. Add the following code just below the line that extracts the coordinates from the arguments list:

    ```
    set len [llength $coords]
    if [expr $len % 2] {
       puts "You must supply an even number of values for coordinates."
       exit
    }
    ```

    The `llength` command returns the length of a list. Remember, any string in Tcl can be treated as a string, in which spaces are characters, or a list, in which at least one space separates one item from the item that precedes it.

    Once you know how long the coordinates list is, you can check that there are an even number of coordinates using the remainder operator “%”. This operation divides the first value by the second and returns only the remainder. The `expr` command is necessary because it instructs the Tcl interpreter to treat a value as a number, not a string.

    This `if` statement makes use of the fact that the condition portion of a Tcl `if` statement can be a numerical value. The number 0 is equivalent to **false** or **no**. All other numbers are equivalent to **true** or **yes**. Thus, the `if` statement translates into “if the remainder of $len/2 is not 0, then print the error message and exit.”

11. Run the script to verify that it works, add comments to make the script more readable, and save it to *no_error.tcl*.

    If this were a real script for production use, you would try to anticipate all possible error conditions and add conditional statements to keep them from causing problems. Things you might consider doing to get more out of this exercise include:

    - Use a `catch` statement with `$L create layer`. If there is an error, warn the user that the second argument must be a number.
    - Use the `string is` command to check that coordinates are actually numbers.
    - Use a `catch` statement with `$L gdsout` in case the program cannot open the file for writing.

Completed Script no_error.tcl

```
##############################################################
# Filename:
#    no_error.tcl
#
# Description:
#
#    Creates a new layout and populates it with a polygon.
#
##############################################################
# This script requires the following arguments:
#  arg 0 --> cell name
#  arg 1 --> layer number
#  args 2 through n --> coordinates for polygon
#                         (minimum of 4 values)
#  arg (n+1) --> file name
#
############################################################

if {$argc < 7} {
   puts "wrong number of arguments"
   puts "<cell name> <layer number> <coords> < file name>"
   exit
}
set L [layout create]
set my_cell [lindex $argv 0]
$L create cell $my_cell
$L create layer [set layer1 [lindex $argv 1]]

#
# get polygon coordinates and create it
#

set first_coord 2
set last_coord [expr $argc - 2]
set coords [lrange $argv $first_coord $last_coord]
set len [llength $coords]
if [expr $len % 2] {
   puts "You must supply an even number of values for coordinates."
   exit
}
eval $L create polygon $my_cell $layer1 $coords

#
# save the layout to the file
#

$L gdsout [lindex $argv end]
```

Script 4: Combining Two Layouts

Topics introduced:

- Multiple layouts
- Creating cell references

For this example, assume you have two test patterns, *tp100-120.gds* and *tp130-150.gds*.

1. Create a Tcl script, called *layout_assembly.tcl* and add the following lines of code:

   ```
   set L1 [layout create tp100-120.gds]
   set L2 [layout create tp130-150.gds]
   ```

   These lines create two layouts, one containing each of the test patterns. Calibre DESIGNrev lets you create as many layouts as you need. The key is storing their handles in variables so you can access the layouts later.

2. Create a third empty layout:

   ```
   set L3 [layout create]
   ```

   You will combine the two test patterns by copying them into this new layout.

3. Create a top cell for the new layout:

   ```
   $L3 create cell MYTOP
   ```

4. Copy the test patterns into the new layout as additional cells (note that Tcl evaluates the [$L topcell] command first):

   ```
   $L3 create cell AA $L1 [$L1 topcell]
   $L3 create cell BB $L2 [$L2 topcell]
   ```

5. Create references to the test pattern cells inside the top cell:

   ```
   $L3 create ref MYTOP AA 0 0 0 0 1
   $L3 create ref MYTOP BB 160000 0 0 0 1
   ```

6. Write the layout to a file.

   ```
   $L3 gdsout temp.gds
   ```

Chapter 5
Writing Macros

Macros are application extensions that you write to add new functionality to the Calibre WORKbench, DESIGNrev, or MDPview applications. One of the many uses for macros is to provide access to the many Calibre batch tools (such as DRC, LVS, the RET batch tools), essentially making them interactive from the graphical user interface. However, you can use macros to perform many other functions as well.

You write macros using the Tcl programming language and the Macros create command. The Macros create command registers the macro with the application. Once registered, the macro name can be placed on the Macros menu as a menu item. The application generates the Macros menu in response to the following application events:

- Startup
- Evaluation of a Macros menu command

When the application generates the menu, it creates a menu item for each currently registered macro.

On installation, the Macros menu contains several Mentor Graphics supplied macros. These macros provide useful functionality and also serve as examples of the sorts of functionality you can add to the application.

This chapter describes how to write functional extensions for your application and how to make the extensions accessible from the **Macros** menu. It is organized into the following sections:

1. How to Write a Macro
2. Exercises in Creating Macros
3. Macro Examples to Study and Learn

Note

User-defined macro extensions involves working with the Mentor Graphics extensions to Tcl/Tk. The proprietary programming objects you will use when creating macros are:

cwbWorkBench Object — the main layout editor widget
cwbLayoutView Object — the view widget for the layout
Layout Object — the Tcl object containing the layout data

How to Write a Macro

Writing macros typically involves three steps:

1. Writing the Command Procedure to invoke another application or perform data manipulation. The macro command procedure must be gui-independent. It relies on the GUI plug-in to pass it the proper user input. In other words, it cannot depend on the state of the application and it cannot query the user for additional information. It cannot contain Tk commands.
2. Writing the GUI Plug-in to define how the macro interacts with the graphical user interface. Typically, the plug-in detects state-dependent information or queries for user input, then invokes the macro command procedure. It can also display data returned by the macro command procedure.
3. Registering the Macro with the **Macros create** command, which also assigns it a name and builds the functional macro from the macro command procedure and the GUI plug-in procedure. Once the macro is registered, you can issue the **Macros menu** command to refresh the Macros menu so that the new command is available to users.

Where To Put Macro Code

The code you write to create a macro can be located in either of the following places:

- Inside the *.signaext* file — If you define the macro in the *.signaext* file, be sure to embed it within a conditional statement that instructs the application to ignore the macro when the Tk software package is not loaded, as described in “Tcl Versus Tk and What the Distinction Means to You” on page 15.
- In a separate file — If you define the macro in a separate file, you will need to load it into the application. Three options for this are:
    - Source the file on invocation, then issue the Macros menu command.
    - Source the file from the shell window, then issue the Macros menu command.
    - Source the file in the *.signaext* file. In this case, you do not need to issue the Macros menu command because the application processes this file on invocation, before it builds the menus.

Writing the Command Procedure

When you are first learning to write macros, it may be easiest if you write the command procedure before you write the GUI plug-in. Writing the procedure first ensures that you know what information you must obtain from the user.

The command procedure performs the actual data manipulation for the macro. You write it using the standard Tcl `proc` format. Note, however, that the procedure must follow these conventions:

- It must be scriptable. That is, once the GUI plug-in invokes it, it requires no user interaction and all processing occurs automatically.
- It must contain no Tk code.
- It can perform operations only on a layout object. That is, it should not depend on the state of the Viewer.

There are no constraints on the arguments for command procedures, nor are there further constraints on the types of processing it can perform. Any valid Tcl script is allowed.

Writing the GUI Plug-in

When the user chooses a macro off the Macros menu, the application calls the GUI plug-in procedure for that macro, passing it two pieces of information: the handle (internal name) for the application, and the pathname of the window from which the macro was invoked. The plug-in procedure uses this information to obtain the data it must pass to the command procedure. Once the command procedure finishes processing, it may or may not return data to the GUI plug-in. Thus, the GUI plug-in performs three functions:

- Obtaining the data
- Invoking the command procedure
- Optionally, displaying a message to the user or updating the GUI the data as necessary to display the new data

Obtaining the Data

You can obtain information to pass to the command procedure two ways:

- Use the handle for the application main widget plus the various methods for the cwbWorkBench Object class (to which the application main widget belongs) to access various sorts of state-dependent data.
- Query the user for additional information needed.

Obtaining Data from the Application

Some or all of the information required to run a macro is already available from the application itself. All it takes to obtain this information is knowing the handle of the WORKBench object, which the application passes to the GUI plug-in.

You can obtain two types of information from the application:

- database information — cells, layers, contents of cells, and so on
- state-dependent information — "current" objects, selections, cursor location, viewer depth, and so on

The examples in Chapter 3 show how you can obtain database information (the handle for a layout, the cells, and the hierarchical relationship between the cells) from the application.

Querying the User for Information

The information that cannot be obtained from the application must come from the user. Querying the user for information can involve either displaying a dialog box requesting information or prompting for information at the command line prompt in the shell. Of the two, displaying a dialog box is the preferred method, because when the macro is waiting for input at the command line, it can look as though the application is hung up.

Displaying the Results of Processing

When the command procedure finishes processing, it returns any results to the GUI plug-in, which must then display the results as necessary.

- Some procedures do not return any data.
- Some procedures return information that needs to be displayed to the user. The easiest way to display information is by using one of the standard Tk widgets such as the messagebox or dialog widgets.
- Some procedures return information indicating that the database, layer panel, or other graphical object needs to be updated. Refer to $cwb updateDisplay for a description of the cwbWorkBench command for updating the GUI.

Registering the Macro

The final step in writing the macro is to create it by assigning it a name and defining how each of the procedures you have written should be used. You do this using the **Macros create** command. Once the macro exists, you add it to the Macros menu.

You can set up the application to load your new macro automatically by:

- Adding all relevant macro code to the *.signaext* file.

- Using the source command in the *.signaext* file to source the Tcl file that contains the macro code.
- Note that if you use either of these methods, you should embed the `source` command in a conditional statement, as described in "Tcl Versus Tk and What the Distinction Means to You" on page 15.

Or you can load the new macro as needed using either of the following methods:

- Calling the file that contains the macro code on invocation, using the -s option on the command line. Since the application evaluates this file after it creates the Macros menu, you must issue the **Macros menu** command in the shell window to make the macro available to the users.
- Sourcing the file that contains the macro code from the shell window, then issuing the **Macros menu** command.

Tip: You can also create a custom keyboard accelerator for user-defined macros. For information, see the section "Custom Key Bindings" in the *Calibre WORKbench Users Manual*.

Exercises in Creating Macros

This section walks you through the process of creating a simple macro that writes the hierarchy of the current layout to a file that the user specifies. This macro builds upon the examples in Chapter 2, which taught you how to write a script to write the hierarchy to the file dump_hier.txt. The macro is similar to the WRITE HIERARCHY macro, which is one of the standard macros available in the GUI. When you have finished writing this macro, you can compare the results it generates to those generated by the WRITE HIERARCHY macro.

- Example 1: Transforming a Simple Script into a Command Procedure
- Example 2: Exploring Ways to Obtain User-Supplied Data
- Example 3: Writing a GUI Plug-in Procedure
- Example 4: Displaying a Message to the User

Example 1: Transforming a Simple Script into a Command Procedure

Topics introduced:

- Requirements for a command procedure
- Sourcing a Tcl script from the shell window

For this example, you modify the file *dump_hier.tcl*, which you created in "Example 2: Using a Recursive Procedure to Write a Layout Hierarchy" on page 41.

The command procedure will perform all the data processing for the macro. After you have written and tested it, you will write the GUI plug-in that calls this procedure.

1. Open *dump_hier.tcl*, from "Example 2: Using a Recursive Procedure to Write a Layout Hierarchy" on page 41 and look for ways to make it more flexible:

Original Script

```
set fileID [open dump_hier.txt w]       a
# Open the layout
set layout [layout create std_.gds]     b
# Dump the hierarchy to the file
dump_hier $layout $fileID
# Close (and save the file)
close $fileID
```

a. **Name of file to open:** For maximum flexibility, change the hard-coded file name into a variable. Later, you will write the GUI plug-in to get the full pathname for the file from the user.

b. **Layout handle:** Macros only work from the GUI, and you can assume that the layout is already opened. Instead of using the create layout command, you can get the handle for the current layout using the $cwb cget command.

1. Modify this portion of the script, then save the modified file as *write_hier.tcl*:

```
      set fileID [open $File w]
      set layout [$cwb cget -_layout]
      dump_hier $layout $fileID
      close $fileID
```

2. By examining the body you have written, you can tell what arguments you need to pass to the procedure. These are `File` and `wb`. Modify the code, using the `proc` command to pass this information to the code you have written:

```
   proc write_hier { wb File} {
      set fileID [open $File w]
      set layout [$cwb cget -_layout]
      dump_hier $layout $fileID
      close $fileID
   }
```

   This command will serve as the command procedure. There are no special requirements for a command procedure, other than that it cannot contain any Tk commands.

3. Add comments to the new procedure to make the file easier to read and update, then save the file again:

```
   ##############################################################
   # PROC: dump_hier
   ##############################################################
   # L      = layout object name
   # F      = file to dump hierarchy to
   # C      = cell to start dumping from, "" means use topcell
   # Indent = string used to format the hierarchy
   ##############################################################
   proc dump_hier { L  F {C ""} {Indent ""} } {
        if {$C == ""} {
          set C [$L topcell]
        }
        puts $F “$Indent --> $C”
        append Indent “        “
       foreach child [$L children $C] {
                            dump_hier $L $F $child $Indent
        }
   }
   ##############################################################
   # COMMAND PROC: write_hier
   ##############################################################
   # Args:
```

```
#     wb     = WorkBench object handle
#     File   = path to file to write to
##############################################################
proc write_hier { wb File} {
   set fileID [open $File w]
   set layout [$cwb cget -_layout]
   dump_hier $layout $fileID
   close $fileID
}
```

4. Invoke Calibre DESIGNrev in interactive GUI mode.

   ```
   calibredrv
   ```

5. Load in the layout of your choice.
6. Switch from the GUI window to the terminal window.
7. Press **Enter** to access the Tcl "%" prompt, then source the file you have just written:

   ```
   source write_hier.tcl
   ```

   The Tcl **source** command passes the contents of the file to the Tcl interpreter. It is equivalent to typing the script directly into the shell window. Notice that because the Tcl file contains only two procedures and no actual commands to evaluate, nothing appears to happen.
8. Get the handle for the application:

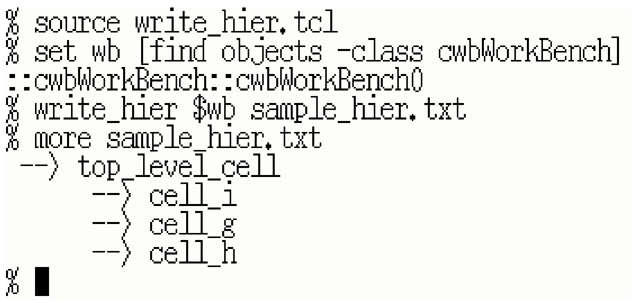
   ```
   set wb [find objects -class cwbWorkBench]
   ```

9. Invoke the write_hier procedure, passing it the handle to the application plus the name of a file to create:

   ```
   write_hier $wb "sample_hier.txt"
   ```

10. View the contents of the file you just created:

    ```
    more sample_hier.txt
    ```



```
% source write_hier.tcl
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% write_hier $wb sample_hier.txt
% more sample_hier.txt
 --> top_level_cell
      --> cell_i
      --> cell_g
      --> cell_h
% 
```

Example 2: Exploring Ways to Obtain User-Supplied Data

Topics introduced:

- The `source` command, continued
- The Tcl `gets` command
- The `tk_getSaveFile` command
- Configuring the file browser widget

1. Create a file called *prompt.tcl* containing the following lines of code:

   ```
   puts "This macro creates a file containing the layout hierarchy."
   puts "Please enter the full pathname for the file to create."
   set filename [gets stdin]
   puts “You entered $filename”
   ```

   This script relies heavily on the Tcl commands `puts` and `gets`:

   - It uses the `puts` command three times: to display a message to the user, to prompt for input, and later, to report back on the data the user entered. Notice that you need not specify a channelID. The default channelID for `puts` is `stdout`, which is the shell window.

   - It uses the `gets` command to capture the data the user enters via the keyboard. The `gets` command does not have a default channelID, so you must tell explicitly where to get the information from. In this case, the information comes from `stdin`, which is the keyboard.

2. Invoke Calibre DESIGNrev in GUI mode.

3. Switch from the GUI window to the terminal window by clicking in the terminal window.

4. Press **Enter**. When you see the “%” prompt, type the following command, then press **Enter**.

   ```
   source prompt.tcl
   ```

   The script you sourced in the previous example contained only procedure definitions, so it did not appear to do anything. When you source this script, it processed the first two lines, displaying the message and the prompt, then waits for your input.

5. Type:

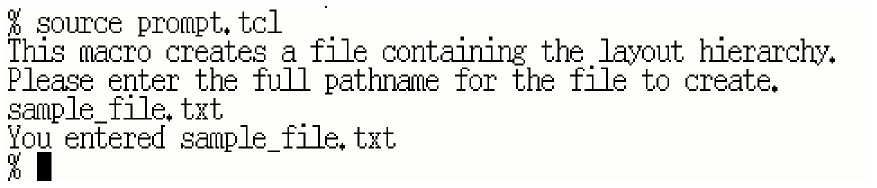
   ```
   sample_file.txt
   ```

   Note that nothing happens until you press **Enter**.

6. Press **Enter**.

The shell window tells you what filename you entered.

Figure 5-1. Capturing Input from the Terminal Window



```
% source prompt.tcl
This macro creates a file containing the layout hierarchy.
Please enter the full pathname for the file to create.
sample_file.txt
You entered sample_file.txt
% 
```

The problem with this method is that it assumes the person using the macro knows to switch to the shell window to view the prompt and enter the data.

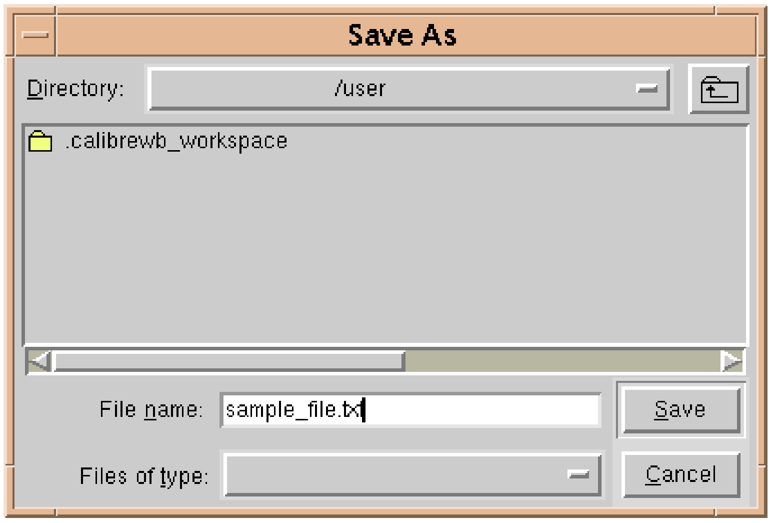
7. Create a new ASCII file called *browser.tcl* containing the following lines of code:

```
set filename [tk_getSaveFile]
puts "You entered $filename"
```

8. Source the script in the terminal window.

```
source browser.tcl
```

Notice that a file browser dialog box pops up.



Save As

Directory: /user

.calibrewb_workspace

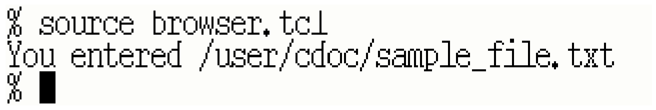
File name: sample_file.txt

Files of type:

The Tk command `tk_getSaveFile` creates a file browser widget. When you enter a file name, either by selecting a file or typing a name in directly, and click Save, the command passes the file name back to the application. Thus, the variable `filename` gets set to the file name the user specifies.

9. Type in the filename “sample_file.txt” and click **Save**.

10. Check the terminal window. The script tells you what filename you entered.



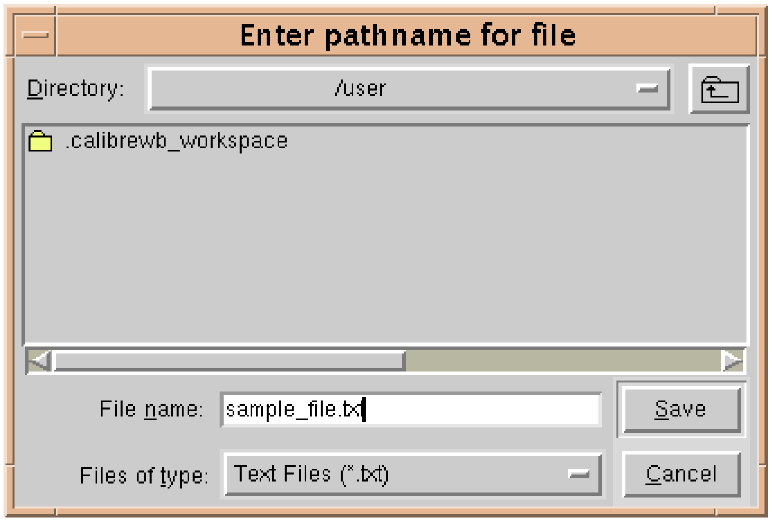
```
% source browser.tcl
You entered /user/cdoc/sample_file.txt
%
```

11. Next, modify *browser.tcl* to match the following code.

```
set title "Enter pathname for file"
set types {
    {{Text Files}       {.txt} }
    {{All Files}        *      }
}
set filename [tk_getSaveFile -filetypes $types -title $title]
puts "You entered $filename"
```

The command arguments for `tk_getSaveFile` lets you configure the file browser widget. In the lines above, you define a title for the widget, and identify types of files the user can display and select from.

12. Save the file, then source it from the Tcl prompt.



Enter pathname for file

Directory: /user

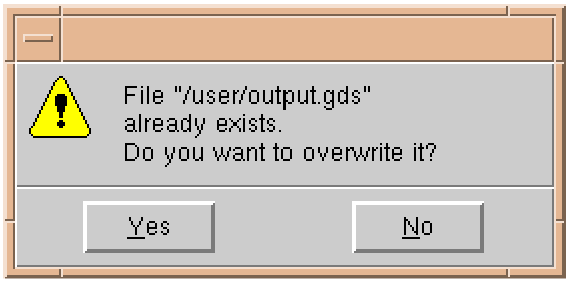
.calibrewb_workspace

File name: sample_file.txt

Files of type: Text Files (*.txt)

13. Navigate to an actual file and select it, then click **Save**.

The software displays a dialog box asking if you want to overwrite the file. One of the added benefits of using the file browser widgets provided by Tk is that they can simplify writing code to manipulate files.


File "/user/output.gds"
already exists.
Do you want to overwrite it?

Example 3: Writing a GUI Plug-in Procedure

\

Topics introduced:

- Arguments to a GUI plug-in
- Invoking the Command Procedure from the GUI plug-in

This example assumes you have Calibre DESIGNrev running in interactive GUI mode with a layout loaded.

1. Open the file *write_hier.tcl* and add the contents of *browser.tcl* to the bottom of the file.

```
set title "Enter pathname for file"
set types {
    {{Text Files}       {.txt} }
    {{All Files}        *      }
}
set filename [tk_getSaveFile -filetypes $types -title $title]
puts "You entered $filename"
```

2. Turn the code from *browser.tcl* into an actual procedure.

```
proc write_hier_plug_in { wbHandle window} {
   set title "Enter pathname for file"
   set types {
    {{Text Files}       {.txt} }
    {{All Files}        *      }
   }
   set filename [tk_getSaveFile -filetypes $types -title $title]
   write_hier $wbHandle $filename
}
```

A GUI plug-in procedure is always passed two arguments: the handle for the WORKbench application, and the handle for the window from which the macro is invoked. You will rarely use the second variable (`window`), but you must include it in the procedure definition; otherwise, the Tcl interpreter will generate an error when it attempts to invoke the macro.

3. Replace the line "`puts "You entered $filename"`" with a call to the new procedure.

```
write_hier $wbHandle $filename
```

This procedure call passes to write_hier the two pieces of information it requires: the handle for the WORKbench application, and the name of the file it is to create.

4. Add comments to ensure that your script is readable by others.

```
###############################################################
# PROC: write_hier_plug_in
###############################################################
# wbHandle = the handle (internal name) for the application
# window = the pathname of the window where the macro was invoked.
###############################################################
```

5. Test the procedure by sourcing the file and calling the GUI plug-in procedure manually:

```
source write_hier.tcl
set wb [find objects -class cwbWorkBench]
write_hier_plug_in $wb any_string
```

   To call the GUI plug-in procedure you must know the handle for the WORKbench application. The Tcl interpreter expects a window handle, as well, but this will not be used, so you can pass it a nonsense string, in this case “any_string.”

6. When the file browser dialog box pops up, enter new_sample.txt, and click Save.

   When the procedure finishes, the shell window displays the % prompt, indicating it is ready for the next command.

7. Exit Calibre DESIGNrev, then open new_sample.txt to see that the procedure actually worked.

Example 4: Displaying a Message to the User

Topics introduced:

- The **tk_messageBox** command
- Using backslash “\” to continue a line

This example assumes you have Calibre DESIGNrev running in interactive GUI mode with a layout loaded.

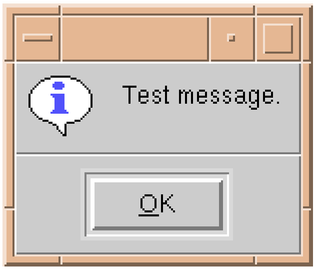
While the write_hier macro does not display any data to the user, it is helpful to display a message box telling them that the macro completed successfully.

1. Experiment with the command that will display the message box by typing it into the shell window:

```
tk_messageBox -message "Test message." -type ok
```

   The `-type` argument defines the buttons to place on this standard widget. There are several different options. OK is the most appropriate one for this situation.

Figure 5-2. The Sample Message Box


Test message.

2. Now add `tk_messageBox` to the write_hier_plug_in procedure. Place it after the call to the write_hier procedure and before the final close bracket.

```
proc write_hier_plug_in { wbHandle window} {
set title "Enter pathname for file"
set types {
  {{Text Files}       {.txt} }
  {{All Files}        *      }
}
set filename [tk_messageBox -filetypes $types -title $title]
write_hier $wbHandle $filename
tk_messageBox -message "The layout hier was successfully written." \
   -type OK
}
```

   Note the backslash "\" at the end of the line beginning with `tk_messageBox`. This indicates that the command would not fit onto one line, and that the line that follows finishes the command. When you use the backslash as a command continuation character, it must not be followed by any other characters. Even a space after it will create an error.

3. Save the file *write_hier.tcl*. and test it by sourcing the file and calling the GUI plug-in procedure manually:

```
source write_hier.tcl
set wb [find objects -class cwbWorkBench]
write_hier_plug_in $wb any_string
```

Example 5-1. Supporting Code

```
##############################################################
# COMMAND PROC: write_hier
##############################################################
# Args:
#     wb     = WorkBench object handle
#     File   = path to file to write to
##############################################################

proc write_hier { wb File} {
   set fileID [open $File w]
   set layout [$wb cget -_layout]
   dump_hier $layout $fileID
   close $fileID
}

##############################################################
# PROC: dump_hier
##############################################################
# L      = layout object name
# F      = file to dump hierarchy to
# C      = cell to start dumping from, "" means use topcell
# Indent = string used to format the hierarchy
##############################################################

proc dump_hier { L  F {C ""} {Indent ""} } {
     if {$C == ""} {
       set C [$L topcell]
     }
     puts $F “$Indent --> $C”
     append Indent “       “
    foreach child [$L children $C] {
                        dump_hier $L $F $child $Indent
     }
}
##############################################################
# PROC: write_hier_plug_in
##############################################################
# wbHandle = the handle (internal name) for the application
# window - the pathname of the window where the macro was invoked.
##############################################################

proc write_hier_plug_in { wbHandle window} {
      set title "Enter pathname for file"
      set types {
        {{Text Files}       {.txt} }
        {{All Files}        *      }
      }
      set filename [tk_getSaveFile -filetypes $types -title $title]
      write_hier $wbHandle $filename
      tk_messageBox -message "The layout hier was successfully written." \
         -type OK
}
```

Putting the Macro on the Menu

Topics introduced:

- The `Macros create` command
- The `Macros menu` command

This example assumes you have Calibre DESIGNrev running in interactive GUI mode with a layout loaded.

1. Add the following command to the end of the file *write_hier.tcl*.

   ```
   Macros create “my macro” write_hier write_hier_plug_in
   ```

   The `Macros create` command requires three arguments:

   - o The name of the macro as it will appear on the Macros menu.
   - o The name of the command procedure
   - o The name of the GUI plug-in procedure

   The arguments must be supplied in the correct order.

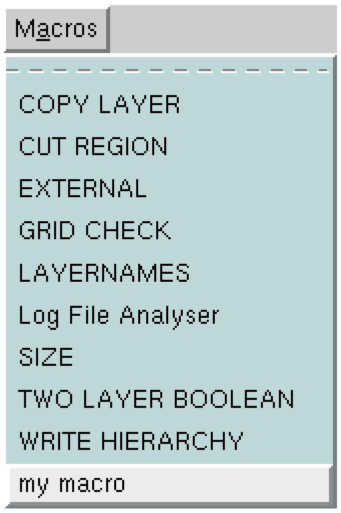
2. Save the file.
3. In the shell window, type the following command, then press Enter.

   ```
   source write_hier.tcl
   ```

4. Also in the shell window, type the macros menu command, then press.

   ```
   Macros menu
   ```

5. Switch to the graphical user interface and click on the Macros menu. You should see your macro displayed there.



Building In Error Handling

- The Tcl `catch` command
- Creating conditional statements using `if` and `if ... else`.
- The Tcl `return` command

This example assumes you have Calibre DESIGNrev running in interactive GUI mode with a layout loaded.

The code you have written so far displays a message box telling you the hierarchy was written successfully to the file. But what if it wasn't? How would you know? How would the program know?

1. In the file *write_hier.tcl*, edit the command procedure write_hier:

   replace the line

   ```
   set fileID [open $File w]
   ```

   with

   ```
   if [catch {open $File w} fileID] {
         tk_messageBox -message "$fileID"}
   ```

   Here you use the `if` command to set up a conditional statement. The basic syntax is:

   ```
   if <condition> <action>
   ```

   If `<condition>` is TRUE, yes, or a non-zero value, then the Tcl interpreter performs `<action>`. Otherwise it skips those actions and moves on to the next portion of the script.

   Since the Tcl `open` command used with the argument `w` opens the file for writing, the command will return an error if you do not have write permission for the file. You can write the `<condition>` based on whether or not you can open the file.

   The `catch` command tracks whether or not a *script*[1] returns an error. If the script executes successfully catch returns 0. If the script does not execute successfully, otherwise it returns an error code. Thus, it makes a perfect condition.

   The `catch` command takes an optional second argument, which is the name of a variable. If the script executes successfully, `catch` sets the variable to the value returned by the command. If the script does not execute successfully, `catch` sets the variable to the error message the script generates. Using this optional argument, you can keep track of the file address, as you did in the original code.

1. The term script refers to a block of code that returns a single value. It can be a single command, a procedure, or a full program.

The `<action>` portion of the **if** command defines what to do if the `<condition>` occurs. If an error occurs, you want to display a message box telling the user of the problem. The easiest way is to display the actual error message, which `catch` saves to the variable fileID. Thus, the `<action>` is:

```
tk_messageBox -message $fileID
```

2. Save *write_hier.tcl* and test it:

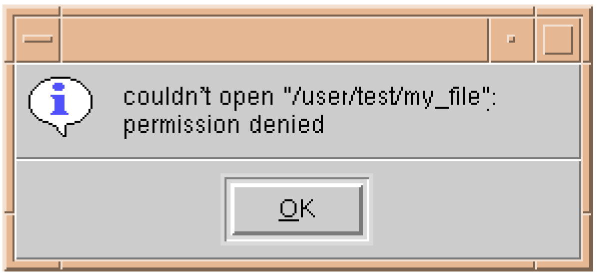
   a. In the shell window, source *write_hier.tcl.*

   b. In the GUI, select “my macro” off the Macros menu.

   c. When the browser pops up, navigate to a file to which you do not have write permission.

   d. Select that file and click Save.

   e. When the macro displays a message box containing the error, click OK.


couldn't open "/user/test/my_file": permission denied

   Before exiting, the macro displays the message box telling you the file was successfully written (which is not true.) Click OK. In the next steps, you will fix this.

3. Edit *write_hier.tcl*, adding the following line inside the <action> for the **if** statement:

```
if [catch {open $File w} fileID] {
      tk_messageBox -message "$fileID"
       return “file_error”}
```

   The Tcl `return` command exits the procedure it is in. It also passes the string you define, in this case “file-error”, back to the program as the *return value*[1] for the procedure.

4. In the file *write_hier.tcl*, edit the command procedure write_hier:

   replace the line:

```
write_hier $wbHandle $filename
tk_messageBox -message "The layout hier was successfully written." \
         -type OK
```

1. The return value for a procedure is the data it returns on completion. By default, the return value is the return value for the last command executed in the procedure. The Tcl return statement lets you control what this is. In many cases, you ignore the return value from a procedure.

with:

```
set return [write_hier $wbHandle $filename]
   if {$return == "file-error"} {
      return
   } else {
      tk_messageBox -message "The layout hier was successfully \
      written”  -type ok
   }
```

The replacement code creates a conditional statement controlling whether or not to the display the message box telling you the layout was successfully written.

First, it uses the `set` command to track the value returned by the write_hier procedure.

It then uses an `if` statement to check what the return value was. If the return value was “file-error” it exits the macro entirely. The `else` portion of this statement defines an `<action>` to perform if the `<condition>` is not met. In this case, the action is to display the macro successful message box.

5. Save *write_hier.tcl* and test it using the process described in step 2. Notice that this time the macro completes after displaying the error message.

Setting up Your Macro to Load Automatically

Topics introduced:

- The order in which the WorkBench application evaluates scripts
- Using backslash “\” to continue a line

This example assumes you have Calibre DESIGNrev running in interactive GUI mode with a layout loaded.

1. Invoke Calibre DESIGNrev in GUI mode, loading in *std_des.gds*.

   ```
   calibredrv -s write_hier.tcl
   ```

2. Check the Macros menu.

   Your new macro is not listed there.When the application starts, it processes the script supplied using -s after it creates the menus.

3. To prove that the application is aware of your macro:

   a. Switch from the GUI window to the terminal window.

   b. Press Enter to access the Tcl “%” prompt, then enter the command to recreate the Macros menu:

      ```
      Macros menu
      ```

   c. In the GUI window, check that your new macro is listed in the Macros menu.

4. Exit the application.

5. Open the configuration file .signaext in your favorite ASCII text editor, and add the following line:

   ```
   source write_hier.tcl
   ```

6. Save the file, then invoke Calibre DESIGNrev:

   ```
   calibredrv
   ```

7. Check the Macros menu.

   Your new macro is listed there. When the application starts, it processes the .signaext file before creating the menus.

Complete Code for the Sample Macro

```
##############################################################
# COMMAND PROC: write_hier
##############################################################
# Args:
# wb = WorkBench object handle
# File= path to file to write to
##############################################################

proc write_hier { wb File} {
   if [catch {open $File w} fileID] {
      tk_messageBox -message $fileID
      return “file-error”
    } else {
      set layout [$wb cget -_layout]
      dump_hier $layout $fileID
      close $fileID
   }
}

##############################################################
# PROC: dump_hier
##############################################################
# L      = layout object name
# F      = file to dump hierarchy to
# C      = cell to start dumping from, ““ means use topcell
# Indent = string used to format the hierarchy
##############################################################

proc dump_hier { L  F {C ““} {Indent ““} } {
     if {$C == ““} {
       set C [$L topcell]
     }
       puts $F “$Indent --> $C”
       append Indent “     “
      foreach child [$L children $C] {
                        dump_hier $L $F $child $Indent
     }
}
##############################################################
# GUI PLUG-IN PROC: write_hier_plug_in
##############################################################
# wbHandle = the handle (internal name) for the application
# window - the pathname of the window where the macro was invoked.
##############################################################

proc write_hier_plug_in { wbHandle window} {
   set title “Enter pathname for file”
   set types {
    {{Text Files}       {.txt} }
    {{All Files}        *      }
   }
   set filename [tk_getSaveFile -filetypes $types -title $title]
   set return [write_hier $wbHandle $filename]
   if {$return == “file-error”} {
      return
   } else {
      tk_messageBox -message “The layout hier was successfully written.”\
        -type ok
   }
}

Macros create “my macro” write_hier write_hier_plug_in
```

Macro Examples to Study and Learn

Source a Tcl Script

```
# ----------------------------------------------------------------------
# SourceTclFile
#
#   Calibre Workbench macro to execute a TCL script
# The following variables are initialized prior to calling the script:
#     $L contains the current layout
#     $C contains the current cell in this layout
#     $V contains the current view
# The layer menu and layout window will be refreshed after the script
# is executed.
# ----------------------------------------------------------------------
proc SourceTclFile {wbHandle window} {
    set types {
      {{TCL scripts} {.tcl}}
      {{all files}     {.*}}
    }
    set s_file [tk_getOpenFile -filetypes $types ]
    if {[file isfile $s_file]} {
        set lname  [file root [file tail $s_file]]
        set L [$wbHandle cget -_layout]
        set V [$wbHandle cget -_layoutView]
        set C [$V cell]
        source $s_file
        $wbHandle updateCWB _dirtylayout  #Update the layout
        $wbHandle updateCWB _dirtyview  #Update the layout
    } else {
        puts "File $s_file does not exist"
    }
}

Macros create {Execute a TCL Script} SourceTclFile SourceTclFile
}
```

List All Layers

In this example, the macro command proc and the macro plug-in are simple to demonstrate the general syntax. You can place the preceding code section in a *.signaext* file to activate the macro. The following Tcl procedure defines a new macro that lists all layers present in the current WORKBench window.

```
# INPUT: wbHandle – Handle to a WorkBench object
# window - Window name for WorkBench object
# OUTPUT: Will print out layers in shown layout.
proc ShowLayersPlugin {wbHandle window}{
      set L [$wbHandle cget -_layout]
      MyShowLayers $L
      }

proc MyShowLayers {layout} {
      puts [$layout layers]
      }

#
# Register the macro
#
Macros create LAYERNAMES MyShowLayers ShowLayersPlugin
```

This simple example follows the required conventions: the plug-in handles any necessary user interaction, and the macro command proc executes the action.

DRC EXTERNAL Spacing Check

This example is more elaborate, and therefore better represents the usefulness of macro extension. This example demonstrates a macro that performs a simple DRC EXTERNAL spacing check. To activate this macro, insert this code into the *.signaext* file.

```
#
# INPUT: wbHandle – Handle to a WorkBench object
# window - Window name for WorkBench object
# OUTPUT: Will print out layers in shown layout.
#
      proc ui_drc_externalcheck {wbHandle window} {
      set L [$wbHandle cget -_layout]
      set lv [$wbHandle cget -_layoutView]
      if { $L == "" } {
         error "Cannot continue without an open layout."

     }
      puts "Hierarchical or flat?"
      set hierflat [gets stdin]
      puts "Check which layer?"
      set layer [gets stdin]
      puts "What distance?"
      set distance [gets stdin]
      puts "What layer for output?"
      set output [gets stdin]
      drc_externalcheck $hierflat $L [$lv cell] \
         [$lv depth] $distance $layer $output
}

proc drc_externalcheck {hierflat handle cell depth distance layers
outputlayer} {

#########################################################
# SETUP SOME BOILERPLATE VARIABLES
#########################################################

set home [set ::env(HOME)]

set inputgds [file join $home .calibrewb_workspace GDS macro_input.gds]

set inputrules [file join $home .calibrewb_workspace tmp macro_rules.svrf]

set outputgds [file join $home .calibrewb_workspace GDS macro_output.gds]

set reportfile [file join $home .calibrewb_workspace tmp macro_output.rep]

set precision [expr int (1/([$handle units]))]

#####################################################
# GENERATE THE SVRF FILE
#####################################################
set buf "//////////////////////////////////////////////\n"
append buf "// INPUT SECTION\n"
append buf "//////////////////////////////////////////////\n"
append buf "LAYOUT SYSTEM GDSII\n"
append buf "LAYOUT PRIMARY \"$cell\"\n"
append buf "LAYOUT PATH \"$inputgds\" \n"
append buf "LAYOUT ERROR ON INPUT YES\n"
append buf [format "PRECISION %d\n" $precision]
append buf "/////////////////////////////////////////////\n"
append buf "// OUTPUT SECTION\n"
append buf "/////////////////////////////////////////////\n"
append buf "DRC MAXIMUM RESULTS ALL\n"
append buf "DRC MAXIMUM VERTEX 199\n"
append buf "DRC RESULTS DATABASE \"$outputgds\" GDSII\n"
append buf "DRC SUMMARY REPORT \"$reportfile\" \n"
append buf "/////////////////////////////////////////////\n"
append buf "// DRC CHECK SECTION\n"
append buf "/////////////////////////////////////////////\n"
append buf "LAYER L1 [lindex $layers 0 ] \n"
append buf "MY_CHECK \{\n\
e1 = EXTERNAL \[L1\] < $distance\n\
EXPAND EDGE e1 BY 0.01\} DRC CHECK MAP MY_CHECK $outputlayer\n"
append buf "/////////////////////////////////////////////\n"
append buf "// DONE \n"
append buf "/////////////////////////////////////////////\n"

# send the buffer to a output function (not documented here)
DumpToFileRef $inputrules buf

####################################################
# GENERATE THE INPUT GDSII FILE
####################################################
if { $hierflat == "hier" } {
$handle gdsout $inputgds $cell
} else {
layout copy $handle tmp_handle $cell 0 $depth
tmp_handle gdsout $inputgds
layout delete tmp_handle
 }

####################################################
# RUN CALIBRE
####################################################
if { $hierflat == "hier" } {
   exec calibre -drc -hier $inputrules >@stdout
} else {
   exec calibre -drc $inputrules >@ stdout

        }

####################################################
# MERGE THE OUTPUT BACK INTO THIS LAYOUT
####################################################
set L2 [layout create $outputgds]
foreach L $outputlayer { $handle create layer $L }
$handle import layout $L2 FALSE append
layout delete $L2

}
# Register this as a macro
Macros create DRC_CHECK drc_externalcheck ui_drc_externalcheck
```

You can merge two separate procedures into a single procedure. An example of this is "Source a Tcl Script" on page 83. However, complex macros are easier to build and debug when created as two separate procedures.

Note that when you use a single procedure to create the macro, you must supply that proc name twice in the Macros create command: Once for the command procedure and once for the gui plug-in procedure. For example:

```
macros create {macro name} my_macro my_macro
```

Loading a File With Layer Colors and Properties

Since files loaded via the layout create command do not have colors loaded, you can use the $cwb loadLayerProperties and $cwb loadDefaultLayerProperties commands to add colors to multiple-layer layouts.

- Create a new Tcl file, and enter the following code:

```
proc loadFile { file {propfile ""} } {
   set W [find objects -class cwbWorkBench]

   # Load the file and any layer properties
   set L [layout create $file]
   layoutSetDefaultColors $L

   # Tell the window about the layer for layer properties loading
   $W configure -_layout $L

   # Load some layer properties
   if {$propfile == ""} {
      # Use the normal layer properties file search
      puts "using normal layer load - $W"
      if { [catch {$W loadDefaultLayerProperties} msg] } {
         puts "ERROR: $msg"
      }
   } else {
      # Load a specified layer properties file
      if { [catch {$W loadLayerProperties $propfile} msg] } {
         puts "ERROR: $msg"
      }
   }

   # Display the layout
   $W viewedLayoutClbk $L
}
```

You can call the script in two ways:

```
% loadFile mix.gds #loads default layer colors
% loadFile mix.gds mix.layerprops # loads named file layer properties
```

Chapter 6
Batch Commands for Layout Manipulation

Table 6-1. Macro Commands

Object	Brief Description
Macros create	Creates a macro.
Macros delete	Removes (un-registers) the specified macro.
Macros menu	Re-creates the **Macros** menu with all registered macros.
Macros show	Returns information about the macros currently registered with the application.

Table 6-2. Database Objects

Object	Brief Description
cwbWorkBench Object	The application.
Layout Object	A layout.
cwbLayoutView Object	A view of the current layout.
StringFeature Object	A set of polygons in the shape of alphanumeric characters that you can place in a layout.
cwbGrid Object	A special object used to manage simulation data.

Table 6-3. Database Commands

	Command	Brief Description
cwbWorkBench	$cwb cget -_layout	Returns the handle of the layout that is visible in the WORKbench main window.
	$cwb cget -_layoutView	Queries the cwbWorkBench object for the handle of the layoutView object.
	$cwb bindKey	Sets a keybinding.
	$cwb loadDefaultLayerProperties	Loads default layer properties.
	$cwb loadLayerProperties	Loads layer properties from a specified file.
	$cwb selectionCloneToNewLayout	Copies all selected geometry to a new layout object.
	$cwb show	Redisplays the layout.
	$cwb rulerMeasurements	Shows information on rulers in the layout.
	$cwb updateDisplay	Redraws the display.
	$cwb viewedLayoutClbk	Changes the currently displayed layout to a specified layout object.
	$cwb zoomAllClbk	Fills the display with the entire layout design.
	$cwb zoomToRectClbk	Zooms the window to the specified coordinates.
cwbGrid	cwbGrid	Creates a grid object.
	$grid dump	Writes the simulation data contained in the grid object to the specified file.

Table 6-3. Database Commands

	Command	Brief Description
layout	layout all	Returns a list of all current layout object handles.
	layout create	Creates a new layout.
	layout copy	Copies data from a source layout into a destination layout, after flattening it to the specified depth.
	layout copy2	Copies all geometry and cell references that touch or are contained within the selection region, preserving any hierarchy that exists.
	layout delete	Deletes the specified layout.
	layout input ascii	Loads a rulecheck results file into a layout.
	layout merge	Merges two layouts, with automatic renaming of the cells to prevent name conflicts.
	layoutSetDefaultColors	Loads system default colors for a layout.
	$L AND	Performs a BOOLEAN AND on the specified layers.
	$L asciiout	Writes the results in the layout to an ASCII file.
	$L bbox	Returns the bounding box of the specified cell.
	$L cellname	Renames the specified cell.
	$L cells	Returns list of all cells.
	$L children	Returns unique children (but not subsequent descendants) of the specified cell.
	$L COMPUTE_AREA	Computes area of objects on a layer.
	$L connect	Specifies connectivity layers.
	$L COPY	Copies the specified layer.
	$L COPYCELL GEOM	Copies the geometries on a given layer in the specified cell.
	$L create cell	Creates a new cell with the specified name.

Table 6-3. Database Commands

	Command	Brief Description
layout	$L create layer	Creates a new layer.
	$L create polygon	Creates a polygon with the given coordinates.
	$L create ref	Creates (or places) a reference of the specified cell.
	$L create text	Creates text as specified.
	$L create wire	Creates a path with the given coordinates.
	$L delete cell	Deletes the specified cell and all references to it.
	$L delete layer	Deletes the specified layer.
	$L delete polygon	Deletes the indicated polygon.
	$L delete polygons	Deletes all geometry from a layer, but leaves the layer in the Layer table.
	$L delete ref	Deletes specified cell reference.
	$L delete text	Deletes the specified text.
	$L delete wire	Deletes the specified path.
	$L exists cell	Checks to see if the specified cell exists in the layout.
	$L exists layer	Checks to see if the specified layer exists in the layout.
	$L expand cell	Flattens all references of the specified cell.
	$L expand ref	Flattens the specified cell references.
	$L file	Returns the filename for the layout.
	$L gdsout	Writes the layout to a GDSII-formatted file.
	$L gridsnap	Depending on the arguments, either snaps all layers and all placements to the specified value of *gridsize* OR checks for violations of a design grid of *gridsize*.
	$L import layout	Imports the specified layout. Options define the manner in which cell name conflicts are resolved.

Table 6-3. Database Commands

	Command	**Brief Description**
layout	$L instancedbout	Writes out instances of the specified cell into ASCII results database.
	$L ismodified	Returns 1 if any modifications were made to the layout since the last save.
	$L iterator (geometries)	Returns a list of specified type of geometries.
	$L iterator (ref | sref | aref)	Returns a list of the specified type of references.
	$L iterator count (geometries)	Returns the number of the specified type of geometries.
	$L iterator count (ref | sref | aref)	Returns the number of the specified type of references.
	$L layerconfigure	Configures a layer with the specified layer properties.
	$L layernames	Returns a list of layer numbers and layer names in the layout.
	The default is -regular.	Returns the given layout's file type setting.
	$L layoutType set	Sets the layout file type.
	$L libname	Sets/gets the GDS LIBNAME variable.
	$L modify text	Deletes an existing text object and creates a new one to replace it.
	$L layers	Returns list of layers in the layout.
	$L NOT	Performs a BOOLEAN NOT on the specified layers.
	$L oasisout	Writes the layout to a OASIS-formatted file.
	$L options	Checks to see which options were set when the layout was created.
	$L OR	Performs a BOOLEAN OR on the specified layers.
	$L query	Returns the nearest object of the specified type.
	$L readonly	Controls whether or not the layout can be saved.

Table 6-3. Database Commands

	Command	**Brief Description**
layout	$L scale	Scales the layout by the given value.
	$L SIZE	Sizes the specified layer.
	$L textout	Returns the text in the specified cell.
	$L topcell	Returns the topcell name.
	$L transcript	Turns recording of transcript information on or off.
	$L transcript OUTPUT	Writes the transcript to an output file.
	$L transcript RANGE	Returns a range of transcript lines.
	$L transcript STATE	Queries the transcript state (whether recording is on or off).
	$L type (deprecated)	Returns the type for this layout.
	$L units database	Sets the size of database units in meters.
	$L units microns	Sets the size of database units in microns.
	$L units user	Sets the size of user units, expressed as a ratio of user units per database units.
	$L update	Updates the layout by recalculating the bounding boxes, topological ordering, and layer tables.
	$L XOR	Performs a BOOLEAN XOR on the specified layers.
StringFeature	StringFeature Object	Creates a stringfeature.
	$str addToLayout	Inserts a stringfeature into an existing layout.

Table 6-3. Database Commands

	Command	**Brief Description**
cwbLayoutView	cwbLayoutView Object	Returns the handle for the current view of the layout.
	$V databaseToScreenUnits	Translates a pair of x and y coordinates from database units to screen units.
	$V depth	Returns the current view depth.
	$V exportView	Outputs a screencapture of the layout viewer.
	$V micronToScreenUnits	Translates a pair of x and y coordinates from microns to screen units.
	$V screenToDatabaseUnits	Translates a pair of x and y coordinates from screen units to database units.
	$V screenToMicrons	Translates a pair of x and y coordinates from screen units to microns.

Several metrology-based Tcl functions are also available to assist in the support of manufacturing machines (metrology machines) with Calibre WORKbench. The Calibre Metrology API (MAPI) is a set of Tcl functions that make possible the rapid development of drivers and analysis tools, and can serve as middleware between tools or data and Calibre analysis programs.

Refer to the *Calibre Metrology API (MAPI) User's Guide and Reference* for a description of the available functions.

cwbWorkBench Object

Within the Tcl/Tk environment, the cwbWorkBench object manages the Calibre WORKbench application. **cwbWorkBench object methods**[1] give you access to information contained in the application. This includes the handle for the current layout and the current view and other state-dependant data.

You reference the cwbWorkBench object by its handle.

- When writing a macro, the cwbWorkBench handle should be considered “known” because the application passes it to the GUI plug-in procedure whenever you invoke a macro.
- When writing scripts or issuing commands at the Tcl prompt, you can get the cwbWorkBench handle using the **find** command:

  ```
  set wb [find objects -class cwbWorkBench]
  ```

- You can also use the **find** command to return multiple handles, which are allocated one per display window (if you have multiple layout viewers open).

In the reference pages that follow, the variable $cwb always refers to the handle of an actual cwbWorkBench object.

$cwb cget -_layout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Queries the cwbWorkBench object for the handle of the layout currently visible in the WORKbench main window. This command returns only one handle, even if you have loaded multiple layouts into WORKbench.

This chapter contains complete list of the layout commands and methods you can use manipulate and operations on layout objects.

Usage

$cwb cget -_layout

Arguments

- **-_layout**

  Required keyword indicating that you are seeking the handle for the layout object, which is a database object.

1. With Tcl, an object is referenced by its handle. The handle also acts as a command for operations performed on the object. Thus, the handle for the cwbWorkBench object becomes the name of the commands you use to access state-dependant data.

Returns

The handle of the current layout.

$cwb cget -_layoutView

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Queries the cwbWorkBench object for the handle of the layoutView object.

Usage

$cwb cget -_layoutView

Arguments

- **-_layoutView**

  Required keyword indicating that you are seeking the handle for the layoutView object, which is a Tk object that controls the display of layout data.

Returns

The handle of the layoutView. If you have created different layoutViews for a single WORKbench session, this command returns the handle of the view that has focus.

$cwb bindKey

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a string consisting of {x1 y1 x2 y2}. This represents either the selected analysis rectangle drawn in the View Center window, or a window defined by the actual viewed window, if no analysis rectangle has been drawn. The default unit for the return data is in microns.

The application supports the key range and key naming format found in Table 6-4. Other Tcl key naming syntax is not recognized by the layout viewer.

Table 6-4. Supported Keyname Formats

Key Type	Key Range
Alt Keys	<Alt-A> thru <Alt-Z>, <Alt-a> thru <Alt-z>
Control Keys	<Control-a> thru <Control-z>, <Control-period>, <Control-comma>
Function Keys	<Key-F1> thru <Key-F12>
Regular Keys	<Key-A> thru <Key-Z>, <Key-a> thru <Key-z>
Special Keys	<Key-KP_Subtract>, <Key-KP_Add>, <Key-Down>, <Key-Left>, <Key-Right>, <Key-Up>

Calibre currently supports custom key binding of the following functions:

Table 6-5. Functions You Can Bind To Keys

Function	Action Performed
cwbCancel	Cancels the the current dialog box.
cwbClearContext	Unsets the selected cell reference as the editing context and returns the editing context to the top cell of the layout.
cwbCloseLayout	Closes the currently active layout. If this was the last layout in the viewer, the window exits.
cwbCloseWindow	Closes the currently active window. If this was the only window open, Calibre exits.
cwbCopy	Copies the selected item to the copy buffer for later pasting.
cwbCreateAref	Opens the Create Array Reference dialog box.
cwbCreateText	Opens the Create Text Reference dialog box.
cwbCreateSref	Opens the Create Scalar Reference dialog box.

Table 6-5. Functions You Can Bind To Keys (cont.)

Function	Action Performed
cwbCut	Cuts the selected item from the layout, copying it to the copy buffer for later pasting.
cwbDecrementEndDepth	Decreases the end depth of the range shown in a hierarchical design.
cwbDecrementStartDepth	Decreases the start depth of the range shown in a hierarchical design.
cwbDeleteSelected	Deletes the selected item.
cwbDuplicate	Duplicates the selected objects, changing the selection focus to the duplicated objects.
cwbDuplicateWindow	Creates a new window with a copy of the currently active layout.
cwbExitApplication	Exits the application.
cwbGoTo	Opens up the Go To dialog box to jump to a specific coordinate location.
cwbHelp	Opens up the documentation.
cwbIncrementEndDepth	Increases the end depth of the range shown in a hierarchical design.
cwbIncrementStartDepth	Increases the start depth of the range shown in a hierarchical design.
cwbLayerBoolean	Opens the Two Layer Boolean dialog box.
cwbLayerCopy	Opens the Layer Copy dialog box.
cwbLayerSize	Opens the Size dialog box.
cwbLoadDefaultLayerProperties	Loads the default layer properties for the layout viewer.
cwbMacrosCutRegion	Runs the CUT REGION macro.
cwbMacrosExecuteTcl	Runs the EXECUTE TCL Script macro.
cwbMacrosExternal	Runs the DRC External macro.
cwbMacrosGridCheck	Runs the GRID CHECK macro.
cwbMacrosLayerBoolean	Runs the TWO LAYER BOOLEAN macro.
cwbMacrosLayerCopy	Runs the LAYER COPY macro.
cwbMacrosLayerSize	Runs the SIZE macro.
cwbObjectProperties	Opens the Object Properties dialog box.
cwbOpenCell	Opens the currently selected cell in the layout viewer.

Table 6-5. Functions You Can Bind To Keys (cont.)

Function	Action Performed
cwbOpenLayout	Opens the Open Layout dialog box.
cwbOpenPrevCell	Switches the layout viewer back to the previously viewed cell.
cwbPanDown	Pans the layout view down by the percentage specified in the preferences.
cwbPanLeft	Pans the layout view left by the percentage specified in the preferences.
cwbPanRight	Pans the layout view right by the percentage specified in the preferences.
cwbPanUp	Pans the layout view up by the percentage specified in the preferences.
cwbPaste	Pastes the contents of the copy buffer to the current location.
cwbPreferences	Opens the Preferences dialog box.
cwbPreferencesView	Opens the Preferences dialog box, switching to the View tab.
cwbPreferencesGrid	Opens the Preferences dialog box, switching to the Grid tab.
cwbPreferencesRuler	Opens the Preferences dialog box, switching to the Ruler tab.
cwbPreferencesPaths	Opens the Preferences dialog box, switching to the Paths tab.
cwbPreferencesMisc	Opens the Preferences dialog box, switching to the Misc tab.
cwbRedo	Repeats an action you have just specified Undo for.
cwbReloadLayout	Reloads the current layout, discarding all changes.
cwbRotate	Rotates the selected item counterclockwise by 90 degrees.
cwbSaveLayout	Opens the Save Layout dialog box.
cwbSelectAllLayers	Selects all layers in the layout list.
cwbSelectRegion	Selects all objects within the drawn selection region based on the selection mode(s) chosen by the user.
cwbSetContext	Sets the selected cell reference as the editing context for editing in place.
cwbSetDepthZero	Sets the current view depth to 0.

Table 6-5. Functions You Can Bind To Keys (cont.)

Function	Action Performed
cwbSetDepth32	Sets the current view depth to 32.
cwbSetDepthMax	Sets the current maximum view depth to 9999.
cwbSetDrawBoxMode	Selects Create Box mode.
cwbSetDrawPathMode	Selects Draw Path mode.
cwbSetDrawPolygonMode	Selects Draw Poly(gon) mode.
cwbSetDrawRulerMode	Selects Draw Ruler mode.
cwbSetMoveMode	Selects Move mode.
cwbSetSelectMode	Selects selection mode.
cwbToggleCellOutline	Toggles the cell outline visibility.
cwbToggleUserGrid	Toggles the current grid visibility.
cwbUndo	Undoes the most recently executed action.
cwbUpdateDisplay	Refreshes the display.
cwbViewBack	Displays the previous view.
cwbViewForth	Displays the next view (must have viewed Back at least once).
cwbZoomAll	Zooms to view the entire layout.
cwbZoomIn	Zooms in a percentage based on the preference settings.
cwbZoomOut	Zooms out a percentage based on the preference settings.

Usage

$cwb bindKey <*keyname*> [*cwbFunction* | *user_proc*] [[null] *menu_name* “*menu_pick*”]

Arguments

- ***<keyname>***

  Is the key combination (as shown in Table 6-4) to customize. Any key combinations not customized remain as their default values. Note that the angle brackets (< >) around the keyname are required.

- *cwbFunction*

  Is the name of a function to execute (the supported list is shown in Table 6-5).

- *user_proc*

  Is the name of a user process to execute (this can be a Tcl script you have also defined and loaded).

- null

  Deletes the specified keybinding. If the keybinding also appeared on a menu, you must also specify the *menu_name* and "*menu_pick*" arguments to remove the keybinding from the menu.

- *menu_name*

  Is the menu (File, Edit, and so on) that contains the item to relabel with the new keybinding (or remove it from if you are deleting a keybinding). When you create a new keybinding, the application also automatically removes the label for the new keybinding from any menu item that previously used it.

  Note that menu actions are independent from custom key bindings, and cannot be changed; this feature only changes key bindings.

- **"***menu_pick***"**

  Is the item on the menu specified in *menu_name* to relabel. Note that the quotes (" ") are required, and *menu_pick* must exactly match how the item appears on the menu.

Examples

```
bindKey <Key-F1> cwbHelp Help "Open User Manual"
```

Binds the F1 key to open the user manual, and adds the "F1" label to **Help** > **Open User Manual**.

```
bindkey <Key-F1> null Help "Open User Manual"
```

Erases the F1 keybinding, including removing the "F1" label from the Help menu (but leaves the "Open User Manual" label in place).

$cwb loadDefaultLayerProperties

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Loads (or reloads) the default layer properties file, located in ~/.calibrewb_workspace/layerprops, or the file specified on the command line with the -dl argument if one was specified.

Usage

$cwb loadDefaultLayerProperties

Arguments

None.

$cwb loadLayerProperties

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Loads (or reloads) the specified layer properties file.

Usage

$cwb loadLayerProperties *filename*

Arguments

- ***filename***

  Is the filename of a layer properties file to load.

$cwb getSelectionRect

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a string consisting of {x1 y1 x2 y2}. This represents either the selected analysis rectangle drawn in the View Center window, or a window defined by the actual viewed window, if no analysis rectangle has been drawn. The default unit for the return data is in microns.

Usage

$cwb getSelectionRect [**dbunits** | <u>**microns**</u>]

Arguments

- **dbunits** | <u>**microns**</u>

  A required keyword indicating the type of units to use for the return data. The default is <u>**microns**</u>.

$cwb selectionCloneToNewLayout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Copies all selected figures to a new layout object and returns the handle to the new layout object.

Usage

$cwb selectionCloneToNewLayout

Returns

The new layout object.

$cwb show

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Redisplays the layout. It raises the window to the front if it is hidden by other windows.

Usage

$cwb show

$cwb rulerMeasurements

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Displays ruler information for rulers in the specified layout. If specified on the command line, it displays all active rulers. The command uses the following format:

```
P1 {x1 y1} P2 {x2 y2} Center {cx cy} Dx dx Dy dy Distance dist
```

Usage

$L rulerMeasurements

Arguments

None

Examples

```
set W ::cwbWorkBench::cwbWorkBench0
$W rulerMeasurements
{P1 {-133684 -14915} P2 {-110130 -11092} Center {-121907 -13004} Dx 23554
Dy 3823 Distance 23862.2347026}
{P1 {-124399 -11228} P2 {-124399 -14983} Center {-124399 -13105} Dx 0
Dy -3755 Distance 3755.0}
```

You can also use this command in looping statements:

```
foreach ruler [$W rulerMeasurements] { puts $ruler }
P1 {-133684 -14915} P2 {-110130 -11092} Center {-121907 -13004} Dx 23554
Dy 3823 Distance 23862.2347026
P1 {-124399 -11228} P2 {-124399 -14983} Center {-124399 -13105} Dx 0
fsDy -3755 Distance 3755.0
```

A more advanced version of the example:

```
foreach ruler [$W rulerMeasurements] {
  array set measurements $ruler
  foreach index [array names measurements]
     { puts "$index     $measurements($index)" }
  set x1 [lindex $measurements(P1) 0]
  set y1 [lindex $measurements(P1) 1]
  puts "(x1,y1): ($x1,$y1)"
}

{P1 {-133684 -14915} P2 {-110130 -11092} Center {-121907 -13004} Dx 23554
Dy 3823 Distance 23862.2347026}
{P1 {-124399 -11228} P2 {-124399 -14983} Center {-124399 -13105} Dx 0
Dy -3755 Distance 3755.0}
Dx 23554
Dy 3823
Center -121907 -13004
P1 -133684 -14915
P2 -110130 -11092
Distance 23862.2347026
(x1,y1): (-133684,-14915)
```

```
Dx 0
Dy -3755
Center -124399 -13105
P1 -124399 -11228
P2 -124399 -14983
Distance 3755.0
(x1,y1): (-124399,-11228)
```

$cwb updateDisplay

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Redraws the display. Use this after using layout modification commands.

Usage

$cwb updateDisplay

$cwb viewedLayoutClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Changes the displayed layout to the layout corresponding to the specified layout handle, then redisplays the layout.

Usage

$cwb viewedLayoutClbk ***layout_handle***

Arguments

- ***layout_handle***

  A required argument specifying the new layout to display.

$cwb zoomAllClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Fills the display with the entire layout design, zooming as needed.

Usage

$cwb zoomAllClbk

$cwb zoomToRectClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Zooms to the specified coordinates in the layout viewer. Coordinates are in layout coordinate space (dbu).

Note

This command replaces the **$cwb zoomToRect** command in earlier versions of Calibre Tools.

Usage

$cwb zoomToRectClbk ***llx lly urx ury***

Arguments

- ***llx lly urx ury***

  A required argument specifying the coordinates (lower left, upper right) to zoom to in the layout.

cwbGrid Object

Tools Supported: Calibre WORKbench, LITHOview

Description

The cwbGrid object class holds simulation data. It is used only with Calibre WORKbench and LITHOview.

You must create a cwbGrid object any time you use batch processing to perform aerial image or cutline2D simulations. The cwbGrid object you create manages the data returned by the simulations.

- **Use the cwbGrid command to** create a cwbGrid object.
- **Use $grid dump to access the data it contains.**

You reference a cwbGrid object by its handle, which is the name you assign to it.

Note
In the reference pages that follow, the variable $grid always refers to the handle of an actual cwbGrid object.

cwbGrid

Tools Supported: Calibre WORKbench, LITHOview

Description

Creates a cwbGrid object. It is used only with Calibre WORKbench and LITHOview.

Usage

cwbGrid *name*

Arguments

- ***name***

  Required argument defining the name of the grid object you are creating. Every grid object must have a unique name, because the name is also used as its handle.

$grid dump

Tools Supported: Calibre WORKbench, LITHOview

Description

Writes the simulation data contained in the grid object to the specified file. It is used only with Calibre WORKbench and LITHOview.

Usage

$grid dump *fileName*

Arguments

- ***fileName***

  Required argument defining the pathname for the file to which the data will be written.

  - If the file does not exist, it will be created.
  - If it does exist, the command overwrites the existing data.

Output File

The output file is an ASCII text file with one line for every sampling point.

For cutlines, each sampling point is represented by:

```
_h(x,y)=I
```

where:

- ***x*** is the index of the sampling point along the cut line
- ***y*** is the index of the focal plane
- ***I*** is the intensity

For example:

```
_h(2,3) = 0.817325
```

reads: The second sampling along the cutline, at the third focal plane in has an intensity of 0.817325.

For aerial images, each sampling point is represented by:

```
_h(x,y)=I
```

where:

- ***x*** is the aerial image grid in the x direction
- ***y*** is the aerial image grid in the y direction
- ***I*** is the intensity

Layout Object

The layout object manages all layout data loaded into the Calibre WORKbench and Calibre LITHOview applications. Manipulating layout data through a script is performed using layout commands and layout object methods:

- **Layout commands** let you manipulate entire layout objects, creating, deleting, merging, and so on.
- **Layout Object methods**[1] let you manipulate the data (or objects) within a layout. This includes creating or deleting cells, polygons, references, and so on.

  You reference the layout object by its handle. You can load and edit or create any number of layout databases within the Calibre DESIGNrev, WORKbench, LITHOview, or MDPview interactive applications, each referenced by a unique handle.

As with all Tcl and Tk objects, the layout handle also acts as a command for operations performed on the layout.

Note
In the reference pages that follow, the variable $L always refers to the handle of an actual layout object.

layout all

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns the handles of all layout objects existing within the application database.

Returns

A list of all layout object handles.

Usage

layout all

1. With Tcl, an object is referenced by its handle. The handle also acts as a command for operations performed on the object. Thus, the handle for the layout you are working on becomes the name of the commands you use to manipulate the layout data.

layout copy

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Flattens data and copies it from a source layout into a destination layout. You define which data to copy by specifying the cell, a selection rectangle, and ***startDepth*** and ***endDepth***.

Note

Most of the arguments for this method are order-dependent. The only way the software can recognize what an argument signifies is by assessing its position in the command string. Because of this, you cannot specify an argument towards the end of the command string without specifying all the arguments that precede it. Thus, while most of the arguments are technically optional, they may be necessary in many situations.

Returns

The handle for the layout object that was created as a result of the copy

Usage

layout copy ***src*** [*dest* [*cell* [*startDepth endDepth* [*x1 y1 x2 y2* [*selMode*]]]]]
[-preserveTextAttributes] [-preserveProperties]

Arguments

- ***src***

  The handle of the layout object to be copied. This is a required argument.

- *dest*

  The handle of the layout object into which the data will be copied. Allowed values are a user-supplied string, which cannot currently be in use as the handle of an existing layout.

  When not specified, the software generates a layout handle for the new layout.

- *cell*

  The name of the cell in the src layout that will be copied. This cell becomes the topcell of the new layout. To copy the entire layout, specify the name of the topcell.

- *startDepth endDepth*

  The hierarchy depth range containing the data to be copied. Prior to copying, the application flattens the data between the *startDepth* and the *endDepth*. Any data above the *startDepth* or below the *endDepth* is not copied.

  - The default value for *startDepth* is 0.
  - The default value for *endDepth* is 1000.

- *x1 y1 x2 y2*

  The coordinates of the rectangular selection area, specified in database units. The first two values define the lower left corner, the second define the upper right corner. When not specified, it copies the entire layout.

- *selMode*

  An optional switch that determines selection mode. Allowed values are 0 | 1.

  - 0 — copies all geometry contained within the selection region.
  - 1 — copies all geometry that touch or are contained within the selection region.

  The default is 0.

- -preserveTextAttributes

  An optional flag used when copying a layout from a GDS file to instruct the software to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the application. The default is to conserve space by not preserving these attributes.

- -preserveProperties

  An optional flag used when copying a layout from a GDS file to instruct the application to preserve GDS geometry/ref property data (PROPATTR & PROPVALUE). The default is to conserve space by not preserving these properties.

layout copy2

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Copies all geometry and cell references that touch or are contained within the selection region to a new layout object, preserving any hierarchy that exists.

Returns

The handle for the layout object that was created as a result of the copy. This is a system generated handle.

Usage

layout copy2 ***src cell x1 y1 x2 y2*** [-preserveTextAttributes] [-preserveProperties]

Arguments

- ***src***

  The handle of the layout object to be copied. This argument is required.

- ***cell***

  The name of the cell in the src layout that will be copied. This cell becomes the topcell of the new layout. To copy the entire layout, specify the name of the topcell.

- ***x1 y1 x2 y2***

  The coordinates of the rectangular selection area, specified in database units. The first two values define the lower left corner, the second define the upper right corner. These coordinates are required.

- -preserveTextAttributes

  An optional flag used when creating a layout from a GDS file to instruct the software to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the application. The default is to conserve space by not preserving these attributes.

- -preserveProperties

  An optional flag used when copying a layout from a GDS file to instruct the application to preserve GDS geometry/ref property data (PROPATTR & PROPVALUE). The default is to conserve space by not preserving these properties.

layout create

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a new layout and returns a new layout object handle.

When no arguments are supplied, this command creates an empty layout. When a filename (or multiple filenames) are supplied, the command creates a layout containing data loaded in from the specified GDS or OASIS file. When specifying a filename, you can also specify a layer mapping to load selected layers and/or map layers and datatypes in the layout file to specific layers within the layout[1] in the application.

Returns

The handle for the new layout object.

Usage

layout create

layout create [*handle*] *fileName* [{-map *L D LMAP*}…[-only]]
  [-dt_expand] [-preservePaths] [-ignoreGDSBoxes] [-preserveTextAttributes]
  [-preserveProperties][-noReport]

layout create [*handle*] -files *fileNameList* [{-map *L D LMAP*}…[-only]]
  [-dt_expand] [-preservePaths] [-ignoreGDSBoxes] [-preserveTextAttributes]
  [-preserveProperties][-noReport]

Arguments

- *handle*

  The handle to assign to the newly created layout handle. When not specified, the software assigns the layout a system-generated handle. The default is “layoutN”, where N is the number of layouts created prior to this one. Thus, for the first layout created, the default handle is layout0.

- *fileName*

  An optional argument specifying the name of one (*fileName* option) or more layout files (-files *fileNameList* option) to load into the newly created layout object. Each layout file must be either a file or a compressed file. If a file is a compressed file, *fileName* must end with an extension of (.z, .Z or .gz) and the gzip program must be accessible to read a compressed file.

- -files *fileNameList*

  For multiple layout files, separate each file in *fileNameList* with a space. Each file is loaded as a separate, concatenated layout, associated with the same layout handle. To merge layouts, use the layout merge command.

1. Refer to the Calibre WORKbench User’s Manual for information on layer map files.

When you create a layout by loading in data from a layout file, the GUI for the application references the layout by this name rather than its handle, however you must use the handle in any layout commands or methods.

- {{-map *L D LMAP*}... *-only*}

  Optional keywords and associated arguments used to map a layout layer/datatype pair in the GDS or OASIS file (*L*, *D*) to a layer in the layout object (*LMAP*, which must be an integer layer number; the option is designed to map a layer datatype to a layer value) when data is read into the new layout. A value of *D* < 0 matches any datatype.

  Each -map keyword maps one layer/datatype pair to a specific layout layer. You can include multiple -map keywords to define more than one mapping.

  If you supply the optional flag -only, the application loads only those layers you explicitly specify with -map. Without this flag, all layers not specified in a -map statement are loaded and assigned the same layer number as in the GDS or OASIS file.

- -dt_expand

  An optional flag used to expand datatypes so that each layer/datatype combination is mapped to a different layout layer. The layer number within the layout is < layer>.<datatype>.

- -preservePaths

  An optional flag used to preserve path definitions when reading layout files. By default, paths are converted to polygons.

- -ignoreGdsBoxes

  An optional flag used when creating a layout from a GDS file to instruct the software to ignore box records when reading the GDS file.

- -preserveTextAttributes

  An optional flag used when creating a layout from a GDS file to instruct the software to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the application. The default is to conserve space by not preserving these attributes.

- -preserveProperties

  An optional flag used when creating a layout from a GDS file to instruct the application to preserve GDS geometry/ref property data (PROPATTR & PROPVALUE). The default is to conserve space by not preserving these properties.

- -noReport

  An optional switch that, when specified, instructs Calibre to skip printing the byte count during a file read as well as the file summary when it finishes the file read. This is useful for procedures that parse the terminal output transcript.

Examples

The following examples create new layouts by loading in data from a GDS file:

Example 1: only read in layer 1.

```
layout create f1.gds -map 1 -1 1 -only
```

Example 2: read in all layers and map layer 1 to 101 and layer 2 to 102.

```
layout create f1.gds -map 1 -1 101 -map 2 -1 102
```

Example 3: read in all layers and map layer 1, datatype 7 to layer 101.

```
layout create f1.gds -map 1 7 101
```

layout delete

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the specified layout.

Returns

No return value.

Usage

layout delete *handle*

Arguments

- ***handle***

  The handle of the layout to be deleted. This a required argument.

layout input ascii

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Reads in the specified ASCII results database into the CWB layout database. By convention, edge clusters loaded into the results database are represented as polygons with propid 224 and property “e”.

- Each RuleCheck is represented as a layer.
- Each RuleCheck text is stored on its RuleCheck layer as a text object.

This command creates a single topcell; the topcell name is obtained from the first line of the ASCII results database.

Returns

No return value.

Usage

layout input ascii *fileName*

Arguments

- ***fileName***

  The name of the results database to read in. If it is not in the current directory, a path (relative or absolute) must be specified in this argument.

layout merge

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Merges the contents of two layouts into a third layout, with automatic renaming of the cells to prevent name conflicts. Cell contents are concatenated/combined according to the mode you select.

Returns

The new layout handle.

Usage

layout merge [*handle_out*] {***handle1***| ***file1***} {***handle2***| ***file2***} ***bump*** [*del1 del2*] [-mode *mode*] [-dt_expand][-preservePaths] [-ignoreGDSBoxes] [-preserveTextAttributes] [-preserveProperties][-autoAlign]

Arguments

- *handle_out*

  An optional argument defining a handle to assigned to the merged layout.

- ***handle1* | *file1***

  The handle or filename of the first layout to be merged. This a required argument.

- ***handle2* | *file2***

  The handle or filename of the second layout to be merged. This a required argument.

- ***bump***

  A required argument defining an integer value added to the layer numbers of each layer in the second layout (*handle2*.) During the merge, data on like-numbered layers in the two layouts are merged. Be sure to specify a large enough value for *bump* to avoid having like-numbered layers in the two layouts.

- *del1 del2*

  Optional flags that control whether the layouts used as input to the merge are deleted as part of the copy. If you include one of these arguments, you must include both. Allowed values for each are 0 | 1.

  - 1 — delete input layout
  - 0 — don't delete input layout

  The default value for each is zero.

- -mode *mode*

  An optional flag/value pair defining how cell name conflicts are resolved when merging. Allowed values are:

    - **append** — Copies the complete contents from the same named cells in both layouts into one cell with that name. All cells from both layouts with unique names appear in the final layout.
    - **overwrite** — Copies the contents of the layout2 cell definition into the resulting layout for same name cells. All cells from both layouts with unique names appear in the final layout.
    - **rename** — Copies both same named cells into the final layout as unique cells with unique names (_WB*x* extension, where *x* is an integer). All cells from both layouts with unique names appear in the final layout.
    - **force_rename** — Copies all cells from both layouts into the final layout as unique cells with unique names (_WB*x* extension, where *x* is an integer). The new name is created by adding the _WBx extension to the original name, where x is an integer. **force_rename** is the default layout merge operation.

- -dt_expand

  An optional flag used to expand datatypes so that each layer/datatype combination is mapped to a different layout layer. The layer number within the layout is < layer>.<datatype>.

  This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- -preserveTextAttributes

  An optional flag used when merging a layout from a GDS file to instruct the software to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the application. The default is to conserve space by not preserving these attributes.

  This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- -preserveProperties

  An optional flag used when merging a layout from a GDS file to instruct the application to preserve GDS geometry/ref property data (PROPATTR & PROPVALUE). The default is to conserve space by not preserving these properties.

  This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- -autoAlign

  An optional flag used only in Calibre MDPView. When specified, Calibre attempts to align two merged layout areas on top of each other when they do not intersect. The default is not to automatically align areas.

Effects of Merging Layouts On Database Units and $L Units Commands

Merging layouts in Calibre results in a common set of measurement values in the combined layout. The behavior and implementation of the database units differs for the types of layout files Calibre accepts:

- **GDS files**

  GDS files can have a variable database unit and user unit, and their measurement metrics are a ratio of database units to user unit. By default, if you create a new GDS file layout in Calibre, the new layout's user unit is set to 1000 database units, and the database unit is set to 1e^{-09} meters, which resolves a default user unit to be 1 micron (1e^{-6} meters) after the ratio is applied.

- **OASIS files**

  OASIS files use the measurement item 'unit', which always has a size of 1 micron. When loading or creating an OASIS file, Calibre sets the database unit to be microns (1e^{-6} meters), with a user unit size of 1e^{6} database units.

You should be aware of the following side effects on the database units when merging layouts that have different database units.

Table 6-6. Conversion Results of a Layout Merge On Database Units

If the Layout File Metrics Are:	DBU
GDS 1 uses the database units as GDS 2	No Change
two OASIS files	No Change
GDS 1 database units are not equal to GDS 2 database units	Scaled
GDS file in microns, OASIS file	No Change
GDS file database units are not in microns, OASIS file	Convert First

where:

No Change: No re-scaling is required. Occurs when the two layouts have the same scale. (OASIS files always use microns for units, therefore they have the same scale).

Scaled: If two GDS files are different sizes and a simple least common denominator cannot be found, Calibre uses a value rounded to the nearest 1e^{-n} value for the smaller database unit value, and scales the larger database unit to that value. For best results, Mentor Graphics recommends that you scale both layouts before merging them using the $L units database, $L units user, and $L units microns commands.

Convert First: When merging files to OASIS, you must convert the database units to microns (using the $L units microns command) before issuing the layout merge command.

layoutSetDefaultColors

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Loads the default system layer colors for the layers in the specified layout handle. This recoloring overrides any default layout properties that were previously loaded.

Usage

layoutSetDefaultColors ***layout_handle***

Arguments

- ***layout_handle***

  Is a layout handle of a currently loaded layout.

Example

```
# load a file
set L [layout create gds/TP.oas]
layoutSetDefaultColors $L
```

$L AND

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Performs a Boolean AND on the specified layers and writes the output to the layer ***Lout***.

The Boolean AND operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L AND *L1in L2in Lout*

Arguments

- ***L1in***

  The name of the first layer to be ANDed. This is a required argument.

- ***L2in***

  The name of the second layer to be ANDed. This is a required argument.

- ***Lout***

  The name of the output layer for the operation. This is a required argument.

$L asciiout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Writes a previously read-in results database out as an ASCII file.

- Layer names are written as RuleCheck names.
- All text objects are written as RuleCheck text lines.
- If more than one cell is present, it will result in a user error.

Usage

$L asciiout *fileName*

Arguments

- ***fileName***

  The name of the file to write to. The filename may include a relative or absolute path.

$L bbox

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns the bounding box of the specified cell, in database units. The first two values define the lower left corner, the second define the upper right corner.

Usage

$L bbox *cellName* [recompute]

Arguments

- ***cellName***

  A required argument specifying the name of the cell whose bounding box is being returned.

- recompute

  An optional argument that when present, instructs the software to recompute and update the bounding box for the specified cell.

$L cellname

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Renames the specified cell.

Usage

$L cellname *oldName newName*

Arguments

- ***oldName***

  The current name of the cell. This is a required argument.

- ***newName***

  The new name for the cell. This is a required argument.

$L cells

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a list of all the cells in the layout $L.

Usage

$L cells

$L children

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns unique children (but not subsequent descendants) of the specified cell. Use with the Tcl `foreach` command for the best results.

Usage

$L children *cellName*

Arguments

- ***cellName***

  The name of the cell whose children are to be returned. This is a required argument.

$L COMPUTE_AREA

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Computes the area of objects on the specified layer. It returns the Area in dbu.

Note that if the specified layer does not exist, the command returns an error.

Usage

$L COMPUTE_AREA ***layer_number***

Arguments

- ***layer_number***

  The layer number of the layer to compute the area of shapes on.

Example

Returns the area of layer 17:

```
layout0 17
4.61781e+10
```

$L connect

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Specifies connectivity layers to be associated with each other. Only original layout layers can be used with this command; the layout viewer does not perform Boolean operations or device extraction.

Usage

$L connect *layer1 layer2* [by *layer3*]

Arguments

- ***layer1***

  The first layer to connect.

- ***layer2***

  The second layer to connect.

- *layer3*

  An optional third layer used to contain shapes that connect the two layers.

$L COPY

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Copies the specified layer to the output layer.

The COPY operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L COPY *Lin Lout*

Arguments

- ***Lin***

  The layer number or <layer number>.<datatype number> designation for the layer to be copied. This is a required argument.

- ***Lout***

  The layer number or <layer number>.<datatype number> designation for the layer that is a copy of ***Lin***. This is a required argument.

$L COPYCELL GEOM

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Copies all geometries on a specific layer in a cell in a source layout into a cell in the target layout.

The COPYCELL GEOM operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L COPYCELL GEOM *srcLayout srcCell srcLayer destCell destLayer*

Arguments

- ***srcLayout***

  The handle of the layout containing the geometry to copy. This is a required argument.

- ***srcCell***

  The name of the cell containing the geometry to copy. This is a required argument.

- ***srcLayer***

  The layer number or <layer number>.<datatype number> designation for the layer containing the geometry to copy. This is a required argument.

- ***destCell***

  The name of the cell into which the geometry will be copied. This is a required argument.

- ***destLayer***

  The layer number or <layer number>.<datatype number> designation for the layer onto which the geometry will be copied. This is a required argument.

$L create cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a new cell with the given name. By default, this method creates an empty cell. If you supply the optional arguments *handle2* and *cell2*, the method creates a cell containing the data contained in *cell2*, which must be in the layout referenced by *handle2*. Used this way, the function acts like an "import cell" function. That is, it creates the named cell, and then copies all geometry (no placements are copied) into the cell.

Usage

$L create cell *cellName* [*handle2 cell2*]

Arguments

- ***cellName***

  The name of the cell to be created. The name cannot conflict with an existing cell name. This is a required argument.

- *handle2*

  An optional argument specifying the handle for the layout that contains *cell2*, whose contents will be copied into the new cell.

- *cell2*

  An optional argument specifying the name of the cell whose contents will be copied into the new cell.

$L create layer

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a new layer with the specified layer number.

Usage

$L create layer *L*

Arguments

- ***L***

  A required argument defining number for the layer to be created. Must be one of the following:

  - An integer in range 0…65535.
  - A pair of integers separated by a period (.). This format is used to represent a layer/datatype pair as <layer>.<datatype>.

$L create polygon

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a polygon on ***layer*** in ***cellName*** with given coordinates. The last coordinates should not be the same as the first coordinates.   Two point polygons are accepted and interpreted as rectangles, for example, 10 10 40 80 are interpreted as 10 10 40 10 40 80 10 80.

Usage

$L create polygon *cellName layer* [-prop *attr string* [G|U]]* ***x1 y1 x2 y2 .... x_n y_n***

Arguments

- ***cellName***

  The name of the cell in which the polygon will be created. This is a required argument.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer on which to create the polygon. This is a required argument.

- -prop *attr string* [G|U]*

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE). More than one -prop argument can be specified.

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

  Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must be sure to include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, you must select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

- ***x1 y1 x2 y2 .... x_n y_n***

  The coordinates of the vertices of the polygon, specified in database units. If only two coordinates are supplied, the software interprets the polygon as a box. At least two coordinates are required.

$L create ref

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a placement of ***refcell*** inside of ***incell*** with the origin located at (***x.y***). The arguments ***mirror***, ***angle***, and ***mag*** define the orientation of the cell reference. Optional arguments can be specified to create an array of cell references.

Usage

$L create ref *incell refcell x y mirror angle mag* [*cols rows xspace yspace*]
  [-prop *attr string* [G|U]] [-force]

Arguments

- ***incell***

  The name of the cell that is the parent cell of the new cell reference. This is a required argument.

- ***refcell***

  The name of the cell that is to be referenced. This is a required argument.

- ***x y***

  The coordinates at which to place the origin of the cell. This is a required argument.

- ***mirror***

  The mirror to apply to the cell reference. Allowed values are 0 | 1. This is a required argument.

  - 0 — do not apply a mirror
  - 1 — mirror across the x axis of the cell

- ***angle***

  The angle of rotation to apply to the cell reference, in a counter-clockwise direction around the cell origin. Allowed values are 0 through 360. This is a required argument.

- ***mag***

  The magnification to apply to the cell reference. This must be a positive number. A value of 1 equates to no magnification. This is a required argument.

- *cols rows xspace yspace*

  A set of optional parameters used to define an array of cell references. *cols* and *rows* define the number columns and rows in the array, *xspace* defines the horizontal spacing between the origins of cells in adjacent columns, and *yspace* defines the vertical spacing between the origins of cells in adjacent rows. Note that these values can be negative to reflect relative positions below or to the left of the array origin.

- -prop *attr string* [G|U]

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE).

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

  Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must be sure to include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, you must select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

- -force

  An optional argument. If specified, this command forces the creation of a reference if the specified cell reference does not exist. It creates an empty cell in the layout corresponding to the referenced cell.

$L create text

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates the indicated text object in the specified cell.

Usage

$L create text *cellName layer x1 y1 text* [*presentation*] [*strans magnification angle*]

Arguments

- ***cellName***

  The name of the cell in which the text object will be created. This is a required argument.

- *layer*

  The layer number or <layer number>.<datatype number> designation for the layer containing the text object. This is a required argument.

- ***x1 y1***

  The coordinates of the center of the new text object, specified in database units. This is a required argument.

- ***text***

  The text string to be displayed. This is a required argument.

- *presentation strans magnification angle*

  Optional text attribute values, usable only if the layout is type GDS and the text has these GDSII compatible attributes associated with it.

  - presentation - Text presentation (font, vertical, horizontal)
  - strans - Instructions for text transformation (reflection, absolute maginification, absolute angle)
  - magnification - Magnification factor (real number)
  - angle - Angular rotation, clockwise in degrees (real number)

$L create wire

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a wire (also called a path) on ***layer*** in ***cellName*** with given coordinates.

Usage

$L create wire *cellName layer width* [-pathtype *type* [*bgnextn endext*n]]
[-prop *attr string* [G|U]] ***x1 y1 x2 y2 .... x_n y_n***

Arguments

- ***cellName***

  The name of the cell in which the polygon will be created. This is a required argument.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer on which to create the polygon. This is a required argument.

- ***width***

  The width of the path to create. This is a required argument.

- ***-pathtype type***

  An optional argument specifying the type of path to create. Allowed values are:

  - 0 — path with no extension at the beginning or end
  - 2 — path with beginning and end extensions equal to 1/2 the path width
  - 4 — custom path, with user supplied values for the beginning and end extensions

  The default type is 0.

- ***bgnextn***

  The length of the beginning extension for the path. Required when the path type is 4.

- ***endextn***

  The length of the beginning extension for the path. Required when the path type is 4.

- -prop *attr string* [G|U]

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE).

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must be sure to include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, you must select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

- ***x1 y1 x2 y2 .... x_n y_n***

  The coordinates of the vertices of the polygon, specified in database units. This is a required argument.

$L customLayerDrawOrder

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Changes the layer drawing order to the specified ordering.

Usage

$L customLayerDrawOrder [*layer_list*]

Arguments

- *layer_list*

  The new layer order, with each layer number separated by spaces. If this optional argument is not specified, the command returns the current layer number ordering.

$L delete cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the cell and all references to that cell.

Usage

$L delete cell *cellName*

Arguments

- ***cellName***

  The name of the cell to delete. This is a required argument.

$L delete layer

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the layer completely.

Usage

$L delete layer *L*

Arguments

- ***L***

  The layer number or <layer number>.<datatype number> designation of the layer to delete.

$L delete polygon

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the polygon described by its layer and coordinates from the specified cell.

Usage

$L delete polygon *cellName layer* [-prop *attr string* [G|U]] ***x1 y1 x2 y2 … x_n y_n***

Arguments

- ***cellName***

  The name of the cell containing the polygon to delete. This is a required argument.

- ***layer***

  The layer number or *layer_number*. *datatype_number* designation for the layer containing the polygon. This is a required argument.

- -prop *attr string* [G|U]

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE).

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

  Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

- ***x1 y1 x2 y2 … x_n y_n***

  The coordinates of the vertices of the polygon to delete. The complete list of coordinates is required.

$L delete polygons

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes all geometry from a specified layer of a specified cell.

Usage

$L delete polygons *cellName layer*

Arguments

- ***cellName***

  The name of the cell containing the polygons to delete. This is a required argument.

- ***layer***

  The layer number or *layer_number*. *datatype_number* designation for the layer containing the polygons to be deleted. This is a required argument.

$L delete ref

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the cell reference or array of references.

Usage

$L delete ref *incell refcell x y mirror angle mag* [*cols rows xspace yspace*]
[-prop *attr string* [G|U]]

Arguments

- ***incell***

  The name of the cell containing the cell to be deleted.

- ***refcell***

  The name of the cell to be deleted.

- ***x y***

  The coordinates of the cell to be deleted.

- ***mirror***

  The mirror applied to the cell reference to be deleted. Allowed values are 0 | 1.

- ***angle***

  The angle of rotation applied to the cell reference.

- ***mag***

  The magnification applied to the cell reference.

- *cols rows xspace yspace*

  A set of optional parameters, required when deleting an array of cell references. Refer to "$L create ref" on page 143 for more information.

- ***-prop*** *attr string* [G|U]

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE).

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

  Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

$L delete text

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the text object described by its layer, coordinates, and text from the specified cell.

Usage

$L delete text *cellName layer x1 y1 text* [*presentation*] [*strans magnification angle*]

Arguments

- ***cellName***

  The name of the cell containing the text object to delete.

- ***layer***

  The layer number or *layer_number*. *datatype_number* designation for the layer containing the text object.

- ***x1 y1***

  The coordinates of the text object to be deleted.

- ***text***

  The text string to be deleted.

- *presentation strans magnification angle*

  Optional text attribute values, usable only if the layout is type GDS and the text has these GDSII compatible attributes associated with it.

  - presentation - Text presentation (font, vertical, horizontal)
  - strans - Instructions for text transformation (reflection, absolute maginification, absolute angle)
  - magnification - Magnification factor (real number)
  - angle - Angular rotation, clockwise in degrees (real number)

$L delete wire

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes the wire (also called a path) defined by the specified properties. (***layer*** in ***cellName*** with given width and coordinates.)

Usage

$L delete wire *cellName layer width* [-pathtype *type* [*bgnextn endext*n]]
[-prop *attr string* [G|U]] ***x1 y1 x2 y2 .... x_n y_n***

Arguments

- ***cellName***

  The name of the cell from which the polygon will be deleted. This is a required argument.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer containing the polygon. This is a required argument.

- ***width***

  The width of the path to delete. This is a required argument.

- ***-pathtype type***

  The type of path to delete. Allowed values are:

  - 0 — path with no extension at the beginning or end
  - 2 — path with beginning and end extensions equal to 1/2 the path width
  - 4 — custom path, with user supplied values for the beginning and end extensions

  The default type is 0.

- *bgnextn*

  The length of the beginning extension for the path. Required when path type is 4.

- *endextn*

  The length of the beginning extension for the path. Required when path type is 4.

- -prop *attr string* [G|U]

  An optional flag plus associated values defining polygon property data (PROPATTR & PROPVALUE).

  For OASIS databases, specifying a value of G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use a value of U to indicate an OASIS user property.

Note that G (GDS) is used as the default, and is the only allowed value for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties will be discarded.

Note

When loading a layout with properties (using the layout create command), you must include the -preserveProperties option, or the properties will be discarded.

When using the Calibre WORKbench GUI, select the **Preserve Text Attributes** checkbox in the **File > Open Layout with Options** dialog box in order to load the stored -prop values.

- ***x1 y1 x2 y2 .... x_n y_n***

The coordinates of the vertices of the polygon, specified in database units. This is a required argument.

$L exists cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Checks to see if the specified cell exists in the layout. Returns 1 if the cell exists, 0 if it does not.

Usage

$L exists cell *cellName*

Arguments

- ***cellName***

  The name of the cell to check for.

$L exists layer

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Checks to see if the specified layer exists in the layout. Returns 1 if the layer exists, 0 if it does not.

Usage

$L exists layer *layer_num*

Arguments

- ***layer_num***

The layer number or <layer number>.<datatype number> designation for the layer to check for.

$L expand cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Replaces all references to a specified cell with the contents of that cell. This is a flattening of the top level of hierarchy within a specified cell. If the cell contains references to other cells, those references are left intact.

The cell definition itself is left intact and not deleted from the design.

Usage

$L expand cell *cellName*

Arguments

- ***cellName***

  The name of the cell to flatten.

$L expand ref

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Flattens the specified reference(s) or array of references. This method replaces a cell reference or array of references with the contents of the cell.

Usage

$L expand ref *incell cellName x y mirror angle mag* [*cols rows xspace yspace*]

Arguments

- ***incell***

  The cell containing the reference.

- ***cellName***

  The name of the cell being referenced.

- ***x y mirror angle mag*** [*cols rows xspace yspace*]

  The location and orientation of the reference. This set of related arguments identifies exactly which reference to flatten. The optional *cols rows xspace yspace* identifies an array of references.

$L file

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns the filename for this layout. If this layout was not created by loading in data from an existing file, this method returns the string “nofile”.

Usage

$L file

$L gdsout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Writes the layout to a GDS-formatted file. Note that this method does not work in read-only mode.

Usage

$L gdsout *fileName* [*cellName*] [-place] [-texttype *texttype*][-map L [*layer* [*datatype*]]] * [-maxPolygonVertices *value*] [-maxPathVertices *value*] [-noEmptyCells [noRefs]]

Arguments

- ***fileName***

  The pathname for the file to which the layout will be written.

- *cellName*

  An optional argument, that when specified, instructs the method to write only that cell and its descendants to the file. When not specified, the software writes all cells to the file.

- -place

  Instructs the software to write ONLY the geometric contents of specified cell. When no cell is specified, it writes the contents of the cell with the most sub hierarchy.

  Without -place, this function writes the entire contents of the specified cell.

- -texttype *texttype*

  A layer datatype to write the output text to. Overrides any datatype mapping that may be present.

- -map L [*layer [datatype*]]*

  An optional keyword and associated arguments used to control which layers are written to the file, and, optionally, to map an application layer to a different layer/datatype in the GDS file. More than one -map argument can be specified.

  Each -map keyword maps one application layout layer to a specific layer/datatype pair. You can use an arbitrary number of optional -map specifications to filter the layers to write and/or remap those layers with the optional map to datatype.

- -maxPolygonVertices value

  If specified, sets a maximum polygon vertex amount that can exist in a single polygon in a layout. If a polygon exceeds the vertex limit, the $L gdsout function segments the polygon until it has a legal amount of vertices. Default is 8191, with a minimum of 3 and a maximum of 8192 for the Calibre reader. (The GDS maximum is 8191; OASIS has no listed maximum, but the listed value for the property is limited to an unsigned integer.)

- -maxPathVertices value

  If specified, sets a maximum path vertex amount that can exist in a single path in a layout. If a path exceeds the vertex limit, it is broken into smaller chains of paths. Default is 1024, with a minimum of 2 and a Calibre reader maximum of 1024. (The GDS maximum is 8191; OASIS has no listed maximum, but the listed value for the property is limited to an unsigned integer.)

- -noEmptyCells [noRefs]

  An optional keyword used to not write empty cell definitions when writing GDS files. If the optional noRefs suboption is also specified, it suppresses the output of references to empty cells as well as the empty cell definitions.

  This option works recursively; if you suppress references within a cell that contained references to only empty cells, the new empty cell that is created is also suppressed along with its references.

$L gridsnap

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

This method manages the grid snapping functionality provided by the application. Its function varies according to the arguments supplied:

- Default (-checkonly not specified) — Snaps the data in all layers and all placements to the specified value of ***gridsize***. This method modifies the layout data, including placements.
- If -checkonly is specified — Checks for geometries and cell references having coordinates (origins or vertices) that do not fall on the grid points for a design grid of the size ***gridsize***. In this mode, any geometry that violates the grid is copied to the *outputlayer*.
- The -c *cellname* option allows you to choose between performing the operation on the full layout or just inside the specified cell.

Usage

$L gridsnap *gridsize* [-checkonly *outputlayer*] [-c *cellname*]

Arguments

- ***gridsize***

  A required argument, specifying the size of the grid to which the layout data will be snapped.

- -checkonly *outputlayer*

  An optional keyword/value pair that instructs the method to check for violations of the grid defined by ***gridsize***. When used, all data that violates the grid is copied to the *outputlayer*.

  - When polygon or text violate the grid, they are copied to *outputlayer* completely.
  - When cell references violate the grid, a rectangle representing the bounding box of the cell reference is created on *outputlayer*.

  The outputlayer itself is not checked.

- -c *cellname*

  An optional argument that signals the application to apply grid snapping or grid checking to the specified cell rather than the entire layout.

$L import layout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Imports the layout specified by the input handle into the layout object $L. You control how this method resolves cell name conflicts through the ***mode*** argument.

Usage

$L import layout {***handle*** **|** ***file***} ***del mode*** [-dt_expand] [-preservePaths] [-ignoreGDSBoxes] [-preserveTextAttributes] [-preserveProperties][-ignoreDuplicateRefs]

Arguments

- ***handle* | *file***

  A required argument specifying the handle or filename of the layout to import into $L. Calibre searches the set of loaded files before searching on the disk.

- ***del* { TRUE | FALSE }**

  A required switch specifying whether to delete the input layout (given by ***handle***) while processing the import. Specifying TRUE conserves memory.

- ***mode***

  A required keyword controlling how the application resolves cell name conflicts. Allowed values are {append | overwrite | rename}.

  - **append** — If the cell already exists, the method appends imported elements to the existing cell.
  - **overwrite** — If the cell already exists, it is deleted and the new version of the cell is used.
  - **rename** — If cell already exists, the imported version gets renamed the extension _WBx, where x is an integer.

- -dt_expand

  An optional flag used to expand datatypes so that each layer/datatype combination is mapped to a different layout layer. The layer number within the layout is < layer>.<datatype>.

  This option is ignored when merging data from a previously loaded layout (using a handle); it is only used if the imported layout is a file.

- -preservePaths

  An optional flag used to preserve path definitions when reading layout files. By default, paths are converted to polygons.

- -ignoreGdsBoxes

  An optional flag used when creating a layout from a GDS file to instruct the software to ignore box records when reading the GDS file.

- -preserveTextAttributes

  An optional flag used when merging a layout from a GDS file to instruct the software to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the application. The default is to conserve space by not preserving these attributes.

  This option is ignored when merging data from a previously loaded layout (using a handle); it is only used if the imported layout is a file.

- -preserveProperties

  An optional flag used when merging a layout from a GDS file to instruct the application to preserve GDS geometry/ref property data (PROPATTR & PROPVALUE). The default is to conserve space by not preserving these properties.

  This option is ignored when merging data from a previously loaded layout (using a handle); it is only used if the imported layout is a file.

- -ignoreDuplicateRef

  An optional flag. If specified, and the exact placements already exist, the command does not append them.

Example

The following object method imports all cells from another layout, and combines them.

```
$L import layout layout2 FALSE append
```

$L instancedbout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Writes out instances of the specified cell into ASCII results database as trapezoids.

Usage

$Linstancedbout ***fileName cellName***

Arguments

- ***fileName***

  The pathname for the ASCII results database file to which the cell data will be written.

- ***cellName***

  The name of the cell whose instance will be written to the ASCII results database file.

$L ismodified

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns 1 if there were any modifications made to the layout since the last save was performed; otherwise it returns 0.

Usage

$L ismodified

$L iterator (geometries)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a list of objects of the indicated type in the indicated cell and layer. This is analogous to the lrange Tcl command.

Usage

$L iterator { **poly** | **wire** | **text** } ***cell layer*** **range** ***first last***

Arguments

- {**poly** | **wire** | **text** }

  A required keyword used to control the type of object being itemized in the list the operation returns. Options are:

  - poly — polygons and boxes
  - wire — GDS paths
  - text — text

- ***cell***

  The name of the cell containing the objects.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer containing the object.

- **range *first last***

  A required keyword and its arguments, used to specify the range of elements to return. The list returned contains all elements in the original list with indices ***first*** through ***last***, inclusive.

  The last argument can be ‘end’ instead of an integer index.

Information Returned

- Polygon — for each polygon, returns:

  { [{properties}] x1 y1 x2 y2 ... xn yn}

  where:

  - {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the polygon has properties associated with it.
  - x1 y1 x2 y2 ... xn yn — ordered list of coordinates of the polygon vertices.

- Wire — for each wire, returns:

  {width pathtype bgnextn endextn [{properties}] x1 y1 x2 y2 ... xn yn} where:

- o width — path width.
- o pathtype — path type values; will be one of {0, 2, 4}.
- o bgnextn endextn — the beginning line-end extent and ending line-end extent for the path.
- o {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the wire has properties associated with it.
- o x1 y1 x2 y2 ... xn yn — ordered list of coordinates of the path vertices.

- Text — For each text object, returns the layer, coordinates, and string. The information is returned in the following format:

  {text x y [presentation [strans magnification angle]]}

  where:

  - o x y — the coordinates of the text.
  - o text — the string displayed as text.
  - o [presentation [strans magnification angle]] — text attribute values, returned only if the layout is type GDS and the text has these attributes associated with it.

Example

To get the first 4 polygon or box objects in cell TOP, on layer 7:

```
% set geomlist [$L iterator geom TOP 7 range 0 3]
```

Note

The “geom” keyword has been deprecated. It is supported for backward-compatibility reasons. Users should start using the poly option in its place.

$L iterator (ref | sref | aref)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a list of references of the indicated type in the indicated cell. This is analogous to the lrange Tcl command.

Usage

$L iterator { **ref** | **sref** | **aref** } ***cell*** **range *first last***

Arguments

- { **ref** | **sref** | **aref** }

  A required keyword used to control which type of references are counted: single references (**sref**), arrays (**aref**), or both (**ref**).

- ***cell***

  The name of the cell containing the objects.

- **range *first last***

A required keyword and its arguments, used to specify the range of elements to return. The list returned contains all elements in the original list with indices ***first*** through ***last***, inclusive. The final argument can also be 'end' instead of an index.

Information Returned

- Sref — for each single reference, returns the coordinates, width, and height of the bounding box of the reference, cell name, coordinates, and orientation. If the layout is a GDS layout, and the reference has properties associated with it, the command also returns those properties. The information is returned in the following format:

  {cell_name x y mirror angle mag [{properties}]}

  where:

  - cell_name, x, y, mirror, angle, mag — name of the referenced cell, and orientation in the design.
  - {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the ref has properties associated with it.

- Aref — for each array of references, returns the coordinates, width, and height of the bounding box of the reference, cell name, coordinates, and orientation, plus array dimensions and spacing. If the layout is a GDS layout, and the reference has properties associated with it, the command also returns those properties. The information is returned in the following format:

  {cell_name x y mirror angle mag cols rows xspace yspace [{properties}]}

  where:

- cell_name, x, y, mirror, angle, mag — name of the referenced cell, and orientation in the design.
- cols rows xspace yspace — array columns and rows, spacing along the x and y directions.
- {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the ref has properties associated with it.

$L iterator count (geometries)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns number of objects of the indicated type in the indicated cell and layer. The geom filter is used to control whether the count includes geometries (polygons and boxes) or text.

Usage

$L iterator count { **poly** | **wire** | **text** } ***cell layer***

Arguments

- {**poly** | **wire** | **text** }

  A required keyword used to control the type of object being counted. Options are:

  - poly — polygons and boxes.
  - wire — GDS paths
  - text — text

- ***cell***

  The name of the cell containing the objects to count.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer containing the objects to count.

Note
The “geom” keyword has been deprecated. It is supported for backward-compatibility reasons. Users should start using the poly option in its place.

$L iterator count (ref | sref | aref)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns number of objects of the indicated type in the indicated cell. The ref filter is used to control whether the count includes single references, arrays, or both.

Usage

$L iterator count { **ref** | **sref** | **aref** } ***cell***

Arguments

- { **ref** | **sref** | **aref** }

  A required keyword used to control which type of references are counted: single references (**sref**), arrays (**aref**) or both (**ref**).

- ***cell***

  The name of the cell containing the references to count.

$L layerconfigure

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Configures a layer with the specified layer properties.

Usage

$L layerconfigure [<u>-regular</u> | -shadow | -all] ***layerNumber*** [*layer_options*]

Arguments

- -regular | -shadow **|** -all

  An optional argument specifying the type of layers to configure. Allowed values are:

  - -regular — configures only regular layers; those that exists in the layout.
  - -shadow — configures only shadow layers: those that no longer exist in the layout but are saved in case they are needed at a later time.
  - -all — configures all layers, regular and shadow.

  The default is <u>-regular</u>.

- ***layerNumber***

  The layer number or <layer number>.<datatype number> designation for the layer to be configured.

- *layer_options*

  The layer property/value pairs to change. Those properties that are not specified remain unchanged. The layer properties you can configure with this command are:

  - -fill — the fill color. Values can be any valid tcl color or the pound sign “#” followed by the RGB color representation.
  - -outline — the outline color. Values can be any valid tcl color or the pound sign “#” followed by the RGB color representation.
  - -outlinewidth — the line width, expressed in microns.
  - -stipple — The fill pattern. Can be one of:
    - clear
    - diagonal_1
    - diagonal_2
    - wave
    - brick
    - circles

  - speckle (formerly gray50)
  - light speckle (formerly gray12)
  - solid
  - @<pathname>
    where pathname is the path to a .xbm file. It should be the full path to the file.
- -visible — A Boolean indicating whether the layer is visible or not.
  - 0 — not visible
  - 1 — visible

$L layernames

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

This method manages layers within $L. Its function varies according to the arguments you supply:

- No arguments — Returns a list of layer number / layer name pairs, with one pair for every layer in the layout.
- Specify a single layer number — Returns the layer name associated with that layer number.
- Specify a single layer number and layer name — Sets the layer name for that layer number to be *layerName*.
- Specify -oasis — Outputs OASIS layer details.

Usage

$L layernames [-shadow | -regular | -all] [*layerNumber* [*layerName*]]

$L layernames -oasis

Arguments

- [-shadow | -regular | -all]

  An optional argument specifying the type of layers the command operates on. Allowed values for config_option are:

  - -regular — operates on only regular layers; those that exists in the layout.
  - -shadow — operates on only shadow layers: those that no longer exist in the layout but are saved in case they are needed at a later time.
  - -all — operates on all layers, regular and shadow.

  The default is -regular.

- *layerNumber*

  The layer number or <layer number>.<datatype number> designation for the layer whose layer name will be returned or set (if *layerName* is specified.)

- *layerName*

  The string to which the layer name for *layerNumber* will be set.

- -oasis

  An optional switch that outputs a list of OASIS layer name details, as described in the OASIS standard. The list format is as follows:

  ```
  {{record_type n-String interval_type bound_a bound_b interval_type bound_a
  bound_b} ...}
  ```

Example

```
% $L layers
1 2 5
% $L layernames 1 POLY
% $L layernames 1
POLY
% $L layernames
1 POLY 2 2 5 5
```

$L layers

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns list of all the layers in this layout.

Usage

$L layers [-shadow | -regular | -all]

Arguments

- [-shadow | -regular | -all]

  An optional argument indicating the type of layers to return in the list.

  - -regular — returns only regular layers; those that exists in the layout.
  - -shadow — returns only shadow layers: those that no longer exist in the layout but are saved in case they are needed at a later time.
  - -all — returns all layers, regular and shadow.

  The default is -regular.

$L layoutType set

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Sets the supplied layout's type property to the supplied type. Note that no type checking is done. You use this command to update Calibre when you save a layout file as a different type (such as a file loaded as GDS, but saved as OASIS).

Usage

$L layoutType set *type*

Arguments

- ***type***

  The new type for the layout, specified as one of gds, oasis, or ascii.

$L libname

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Retrieves the GDS LIBNAME value if no library name is specified; sets the GDS LIBNAME value if one is specified as an argument.

Usage

$L libname *library_name*

Arguments

- *library_name*

  An optional argument specifying the new value for GDS LIBNAME.

$L modify text

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Deletes an existing text object and creates a new one to replace it.

Usage

$L modify text *cellName layer x1 y1 text cellNameNew layerNew x1New y1New textNew*

Arguments

- ***cellName***

  The name of the cell in which the original text object exists. This is a required argument.

- ***layer***

  The layer number or <layer number>.<datatype number> designation for the layer containing the original text object. This is a required argument.

- ***x1 y1***

  The coordinates of the center of the original text object, specified in database units. This is a required argument.

- ***text***

  The text string displayed by the original text object. This is a required argument.

- ***cellNameNew***

  The name of the cell in which the new text object will be created. This is a required argument. (Not available in Calibre DESIGNrev).

- ***layerNew***

  The layer number or <layer number>.<datatype number> designation for the layer containing the new text object. This is a required argument.

- ***x1New y1New***

  The coordinates of the center of the new text object, specified in database units. This is a required argument.

- ***textNew***

  The text string to be displayed by the new text object. This is a required argument.

$L NOT

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Performs a Boolean NOT on the specified layers and writes the output to the layer ***Lout***.

The Boolean NOT operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L NOT *L1in L2in Lout*

Arguments

- ***L1in***

  The name of the first layer to be NOTed.

- ***L2in***

  The name of the second layer to be NOTed.

- ***Lout***

  The name of the output layer for the operation. Note that the output layer specified should not also be an input layer.

$L oasisout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Writes the layout to a OASIS-formatted file. Note that this method does not work in read-only mode.

Usage

$L oasisout *fileName* [*cellName*] [-place] [-map L [*layer [datatype*]]]
[-maxPolygonVertices *value*] [-maxPathVertices *value*] [-noEmptyCells noRefs][-cBlocks]

Arguments

- ***fileName***

  The pathname for the file to which the layout will be written.

- *cellName*

  An optional argument, that when specified, instructs the method to write only that cell and its descendants to the file. When not specified, the software writes all cells to the file.

- -place

  Instructs the software to write output only the geometric contents of specified cell. When no cell is specified, it defaults to the cell with the most sub hierarchy.

- -map L [*layer [datatype*]]

  An optional keyword and associated arguments used to control which layers are written to the file, and, optionally, to map an application layer to a different layer/datatype in the OASIS file.

  Each -map keyword maps one application layout layer to a specific layer/datatype pair. You can use an arbitrary number of optional -map specifications to filter the layers to write and/or remap those layers with the optional map to datatype.

- -maxPolygonVertices *value*

  If specified, sets a maximum polygon vertex amount that can exist in a single polygon in a layout. If a polygon exceeds the vertex limit, the $L oasisout function segments the polygon until it has a legal amount of vertices. Default is 8191, with a minimum of 3 and a maximum of 8192. (The GDS maximum is 8191; OASIS has no listed maximum, but the listed value for the property is limited to an unsigned integer.)

- -maxPathVertices *value*

  If specified, sets a maximum path vertex amount that can exist in a single path in a layout. If a path exceeds the vertex limit, it is broken into smaller chains of paths. Default is 1024, with a minimum of 2 and a maximum of 1024. (The GDS maximum is 8191; OASIS has no listed maximum, but the listed value for the property is limited to an unsigned integer.)

- -noEmptyCells [noRefs]

  An optional keyword used to not write empty cell definitions when writing OASIS files. If the optional noRefs suboption is also specified, it suppresses the output of references to empty cells as well as the empty cell definitions.

  This option works recursively; if you suppress references within a cell that contained references to only empty cells, the new empty cell that is created is also suppressed along with its references.

- -cBlocks

  Writes CBLOCKS (compression cell blocks) at the cell level. (In the layout viewer, this option is available in the **File > SaveAs** dialog box.)

$L options

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Checks to see the options that were set when the layout was created (using layout create).

- If no argument is specified, returns a list of options set on the layout.
- If an argument is specified, returns a value of 0 or 1 (-map returns a string):
  - 0 — the option was not used when the layout was created.
  - 1 — the options was used when the layout was created.

Usage

$L options [-map | -only | -dt_expand | -ignoreGdsBoxes | -preservePaths | -preserveTextAttributes | -preserveProperties]

Arguments

- [-map | -only | -dt_expand | -ignoreGdsBoxes | -preservePaths | -preserveTextAttributes | -preserveProperties]

  An optional argument specifying the option to report on. At most one option argument is allowed.

  - -map — Returns the map options specified for the layout, or the null string (““) if no mapping options were specified when the layout was created. The map options are returned in a list that has the following format:

```
{layer0 datatype0 mapped_layer0 ... layer_n datatype_n mapped_layern}
```

  - -only — Reports on whether or not the -only (load mapped layers) option was specified when the layout was created.
  - -dt_expand — Reports on whether or not the -dt_expand option was specified when the layout was created.
  - -ignoreGdsBoxes — Reports on whether or not the -ignoreGdsBoxes option was specified when the layout was created. This option is valid only for layouts to be saved in GDS format.
  - -preservePaths — Reports on whether or not the -preservePaths option was specified when the layout was created.
  - -preserveTextAttributes — Reports on whether or not the -preserveTextAttributes option was specified when the layout was created. This option is valid only for layouts to be saved in GDS format.

- -preserveProperties — Reports on whether or not the -preserveProperties option was specified when the layout was created. This option is valid only for layouts to be saved in GDS format.

$L OR

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Performs a Boolean OR on the specified layers and writes the output to the layer ***Lout***.

The Boolean OR operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L OR *L1in L2in Lout*

Arguments

- ***L1in***

  The name of the first layer to be ORed.

- ***L2in***

  The name of the second layer to be ORed.

- ***Lout***

  The name of the output layer for the operation.

$L query

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Selects the requested element nearest to the given (X,Y) coordinate and returns relevant information about that element. The information returned varies according to the type of object.

If more than one object of the same type exists at the specified coordinates, the function selects all the objects and returns a list of object properties.

To perform the search, the application creates a search area in the shape of a box, with (X,Y) located at the center of the box. You can define the size of the box through the optional keywords maxSearchDistanceX (width) and maxSearchDistanceY (height). By default, the box is 1 micron square.

The input coordinates and output coordinates are all be in database units.

Usage

$L query { **ref** | **polygon** | **wire** | **edge** | **vertex**| **closept** | **text** } ***cell startDepth endDepth X Y***
[*maxSearchDistanceX maxSearchDistanceY*]

Arguments

- { **ref** | **polygon** | **wire** | **edge** | **vertex**| **closept** | **text** }

  A required keyword used to control which type of object is being returned. Options are:

  - ref — returns information about the nearest cell reference(s) If the reference is an array, the information also lists the array dimensions and spacing. When more than one cell reference share the same closest point, query returns a list containing all these cell references.
  - polygon — returns the polygon layer and vertices, followed by information about a closest point on the polygon. When more than one polygon share the same closest point, query returns a list containing all these polygons.
  - wire — returns path attributes and vertices, followed by information about a closest point on the path. When more than one wire share the same closest point, query returns a list containing all these wires.
  - edge — returns a nearest edge of a visible polygon.
  - vertex — returns a nearest vertex of a visible polygon.
  - closept — returns the nearest point of a visible polygon. When more than one visible polygon has the same nearest point, **query closept** keeps only the first one that it found.
  - text — returns the nearest text object. When more than one text object share the same closest point, query returns a list containing all these text objects.

- ***cell***

  A required argument identifying the cell in which to search.

- ***startDepth endDepth***

  A pair of required arguments defining the levels of hierarchy to search. The query search only returns objects that exist on a level between *startDepth* and *endDepth*, inclusive.

- ***X Y***

  A pair of required arguments defining the (X,Y) coordinates of the search point in dbu.

- *maxSearchDistanceX maxSearchDistanceY*

  A pair of optional arguments defining the size of the search area. maxSearchDistanceX defines the width of the search area. maxSearchDistanceY defines the height. When not specified, these values default to 1 micron.

Returns

- ref — for each reference, returns the coordinates, width, and height of the bounding box of the reference, cell name, coordinates, and orientation, plus array dimensions and spacing if the reference is an array. If the layout is a GDS layout, and the reference has properties associated with it, the command also returns those properties. The information is returned in the following format:

  {llcx llcy xlen ylen cell_name x y mirror angle mag [cols rows xspace yspace] [{properties}]}

  where:

  - llcx and llcy — lower left corner of the bounding box of the reference.
  - xlen and ylen — width and height of the bounding box of the reference, respectively.
  - cell_name, x, y, mirror, angle, mag — name of the referenced cell, and orientation in the design.
  - cols rows xspace yspace — array information, returned only if the reference is an array.
  - {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the ref has properties associated with it.

- polygon — for each polygon, returns the layer and vertices, followed by information about the closest point on the polygon. If the layout is a GDS layout, and the polygon has properties associated with it, the command also returns those properties. The information is returned in the following format:

  {layer [{properties}] x1 y1 x2 y2 ... xn yn} idx "v"|"e" dist {xpt ypt}}

  where:

  - layer — layer polygon is on.

- {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the polygon has properties associated with it.
- x1 y1 x2 y2 ... xn yn — ordered list of coordinates of the polygon vertices.
- idx — index of the vertex or edge containing the nearest point.
- “v”| “e” — “v” if the closest point is a vertex, “e” if the closest point is on an edge.
- dist — distance between the closest point and the query point.
- {xpt ypt} — coordinates of the closest point.

- wire — for each wire, returns the path attributes and vertices, followed by information about the closest point on the path. If the layout is a GDS layout, and the wire has properties associated with it, the command also returns those properties. The information is returned in the following format:

  {layer width pathtype bgnextn endextn [{properties}] x1 y1 x2 y2 ... xn yn idx "v"|"e" dist {xpt ypt}}

  where:

  - layer — layer path is on.
  - width — path width.
  - pathtype — path type values will be one of {0, 2, 4}.
  - bgnextn endextn — the beginning line-end extent and ending line-end extent for the path.
  - {properties} — a list of property attribute name / value pairs, returned only if the layout is type GDS and the wire has properties associated with it.
  - x1 y1 x2 y2 ... xn yn — ordered list of coordinates of the path vertices.
  - idx — index of the vertex or edge containing the nearest point.
  - “v”| “e” — “v” if the closest point is a vertex, “e” if the closest point is on an edge.
  - dist — distance between the closest point and the query point.
  - {xpt ypt} — coordinates of the closest point.

- closept — returns the nearest point on the nearest visible polygon. Information is returned in the following format:

  {layer x y [x1 y1 x2 y2]}

  where:

  - layer — the layer the polygon is on.
  - x y — the coordinates of the nearest point.

   - [x1 y1 x2 y2] — the coordinates of the vertices of the nearest edge. This information is returned only if the nearest point is not a vertex of the polygon.

  Note that if more than one visible polygons have the same nearest point, then query closept keeps only the first one that it found.

- Vertex — returns information about the nearest vertex on the nearest polygon. The information is returned in the following format:

  {{layer x1 y1 x2 y2 ... xn yn} vidx}

  where:

   - layer — the layer the polygon is on.
   - x1 y1 ... xn yn — an ordered list of all the polygon vertexes.
   - vidx — the index of the vertex in question, starting from x1, y1.

- Edge— returns information about the nearest edge on the nearest polygon. The information is returned in the following format:

  {{layer x1 y1 x2 y2 ... xn yn} eidx}

  where:

   - layer — the layer the polygon is on.
   - x1 y1 ... xn yn — an ordered list of all the polygon vertexes.
   - eidx — the index of the edge in question, starting from x1, y1 as the first vertex of the first edge.

- Text — For each text object, returns the layer, coordinates, and string. The information is returned in the following format:

  {layer x y text [presentation [strans magnification angle]]}

  where:

   - layer — the layer the text is on.
   - x y — the coordinates of the text.
   - text — the string displayed as text.
   - [presentation [strans magnification angle]] — text attribute values, returned only if the layout is type GDS and the text has these attributes associated with it.

$L readonly

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Controls whether or not the layout can be saved.

Usage

$L readonly { **1** | **0** }

Arguments

- **0**

  Layout is not read-only, and can be saved.

- **1**

  Layout is read-only.

$L scale

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Scales the layout by the given value. Due to the effects of grid snapping, scaling is reversible only in the following situations:

- scaling up by an integer value.
- scaling down by an integer N, when all data is known to be at coordinates that are multiples of N.

For all other situations, this operation is irreversible.

Usage

$L scale *F*

Arguments

- ***F***

  The value by which the layout will be scaled.

$L SIZE

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Sizes the specified layers and writes the output to the layer ***Lout***.

The SIZE operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L SIZE *Lin Lout* BY *val*

Arguments

- ***Lin***

  The name of the layer to be sized.

- ***Lout***

  The name of the output layer for the operation.

- **BY *val***

  A required keyword/argument pair used to specify the amount by which the layer is sized. Sizing is performed on a cell-by-cell basis. Hierarchical interactions are not accounted for.

$L textout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns all text elements in the given cell as a string, formatted as follows:

```
{{ layer x y text} ... { layer x y text}}
```

Usage

$L textout *cellName*

Arguments

- ***cellName***

  The name of the cell containing the text to output.

$L topcell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

A topcell is a cell that is not referenced within any other cells. This method returns the name of the topcell in $L. If $L contains multiple topcells, the method returns the one with the most total descendants.

When the all option is specified, this object returns all topcells.

Usage

$L topcell [all]

Arguments

- all

  An optional argument used to return the names of all cells without references to them.

$L transcript

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Turns recording of transcript information on or off. Use with $L transcript OUTPUT, which writes the transcript to a file.

Usage

$L transcript { **1** | **0** }

Arguments

- **0**

  Turns transcript recording off.

- **1**

  Turns transcript recording on.

$L transcript OUTPUT

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Writes the transcript to an output file. Used with $L transcript, which toggles recording.

Usage

$L transcript OUTPUT *fileName*

Arguments

- ***filename***

  The pathname for the file to which the transcript will be written.

$L transcript RANGE

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns a range of transcript lines.

Usage

$L transcript RANGE *lo hi*

Arguments

- ***lo***

  Defines the first (lowest numbered) line.

- ***hi***

  Defines the last (highest numbered) line returned.

$L transcript STATE

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Queries the transcript state (whether recording is on or off).

Usage

$L transcript STATE

Return Values

1 — Transcript is being recorded.

0 — Transcript is not being recorded.

$L type (deprecated)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

This command has been deprecated; use The default is -regular. instead.

$L units database

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Sets or returns the size of database unit.

Usage

$L units database [*val*]

Arguments

- *val*

  An optional argument used to set the size of the database unit, in meters.

  There is no default value for this command. When not specified, the command simply returns the size of the database unit.

  Note that when you create a new database using batch commands, it is defined by default with a database unit size of 10^{-9}, which corresponds to 1 nm. Because the database is also created with a ratio of user units per database unit of .001, this corresponds to a user unit size of 1 micron ($10^{-9}/.001 = 10^{-6}$, which is 1 micron).

Automatic Re-Setting of $L units database

The value of $L units database will be automatically re-set if you issue the $L units microns command.

$L units microns

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Simplifies setting $L units database and $L units user when the user unit size is 1 micron. If no argument is specified, returns the current number of database **units per microns**.

Note

This command is most convenient for use with OASIS databases, for which the user unit size is 1 micron.

For GDS databases, if you need the user unit size to be something other than 1 micron, you should set $L units database and $L units user separately.

You can interpret this command as "database units per micron." The command does the following:

1. Defines the size of the database unit in terms of database units per micron.

   It does this by issuing the command "$L units database *(*10^{-6}/*units_per_micron)":*

   If $L units database = x
   1 micron = 10 $^{-6}$ **meters**
   1 micron = x dbunits
   1 dbu = 10 $^{-6}$/x **meters**

2. Defines the ratio of user units per database unit to be 1/*units_per_micron*.

   It does this by issuing the command "$L units user (1/*units_per_micron*)". Combined with "$L units database (10^{-6}/*units_per_micron*)", this results in a user unit size of 1 micron.

Usage

$L units microns [*units_per_micron*]

Arguments

- *units_per_micron*

  The number of database units per micron.

  There is no default value for *units_per_micron, the argument*. When not specified, the command simply returns the current number of database **units per microns**.

  Note that when you create a new database using batch commands, it is defined such that there are 1000 database units per micron, that is, the database unit is one nanometer.

Examples

A) Issuing the command with an OASIS database:

```
$L units microns 5
```

- Issues the command $L units database (10 $^{-6}$/5)

  or $L units database .0000002.

  How this works:

  1 micron = 10 $^{-6}$ **meters**
  There are 5 database units per micron.
  Each database unit = 1/5 micron.
  1 database unit = 10 $^{-6}$/5 **meters = .0000002**

B) Issuing the command with a GDS database:

```
$L units microns 5
```

- Issues the command $L units user .2.
  This translates into .2 user units in a database unit, or 1 user unit equals 5 database units.
  Since the database unit is .2 microns, the user unit is 1 micron.
- Issues the command $L units database (10 $^{-6}$/5) or $L units database .0000002.
  This translates into .20 micron because 10 $^{-6}$ is one micron, expressed in meters.

$L units user

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Sets or returns a ratio representing the number of user units per database unit[1].

Note

For OASIS files this command is not recommended. It is much more intuitive to use the $L units microns command because the user unit size is always assumed to be 1 micron.

Usage

$L units user [*val*]

Arguments

- *val*

  An optional argument used to define the ratio of user units per database unit for the layout.

  There is no default value for this command. When not specified, the command returns the ratio of user units per database unit.

  Note that when you create a new database using batch commands, it is defined such that the ratio of user units per database unit is .001. Because the database is also created with a database unit size of 10^{-9}, this corresponds to a user unit size of 1 micron.

Differences Between GDS and OASIS with Respect to Units

In GDS databases, you define the physical size of a database unit in meters. *Changing $L units user changes the size of the user unit.*

In an OASIS database, the size of a user unit is fixed at 1 micron. *Changing $L units user without also changing the size of the database unit can result in data that appears corrupted. The best way to avoid problems with data is by using the* $L units microns *command instead.*

Understanding User Units, Precision, and Grids

The Precision statement in SVRF defines the database precision in terms of database units per user unit. The default is 1000. The **$L units user batch command also defines the database precision, but in terms of user units per database unit. N**otice this is the inverse of PRECISION**.**

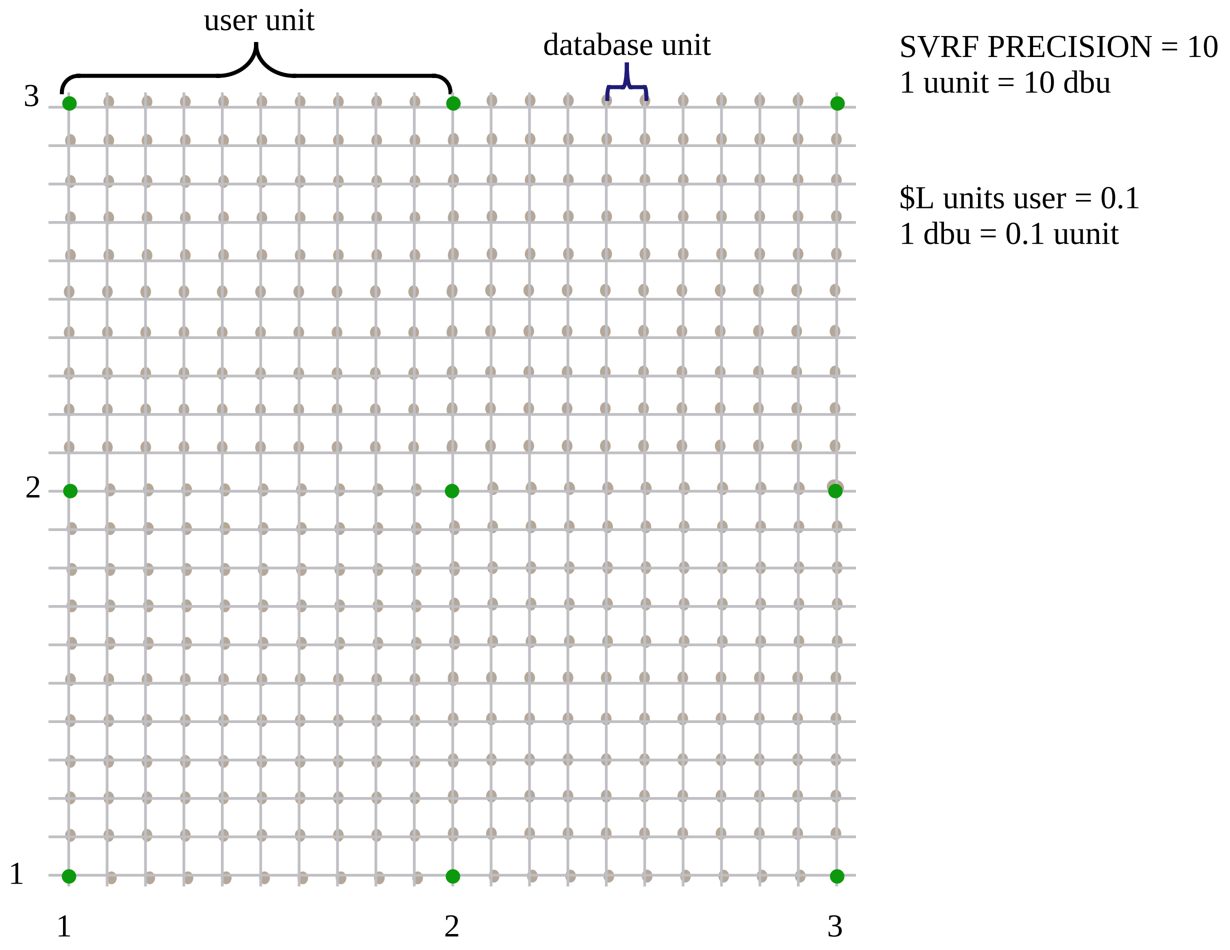
PRECISION and **$L units user** are important because they define the grid to which data is snapped:

- **SVRF PRECISION** = The number of grid points from one user unit to the next.
- **$L units user** = The distance between grid points expressed in user units.

1. Another way to say this is that it defines the size of a database unit, expressed in user units.

This is as shown in Figure 6-1.

Figure 6-1. Relating User and Database Units to the Design Grid



$L update

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Updates the layout by recalculating the bounding boxes, topological ordering, and layer tables.

Usage

$L update

$L viacell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Specifies that the named cell is a via cell. Instances of via cells specified in the layer properties file will be processed flat during net extraction if connectivity is set to Top.

Usage

$L viacell *cellname*

Arguments

- ***cellname***

  The name of the cell to be set as a via cell. Specifying an asterisk (*) in the cell name treats the string as if it contained the wildcard character.

$L XOR

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Performs a Boolean XOR on the specified layers and writes the output to the layer ***Lout***.

The Boolean XOR operation is performed on a cell-by-cell basis. It is not a hierarchical operation.

Usage

$L XOR *L1in L2in Lout*

Arguments

- ***L1in***

  The name of the first layer to be XORed.

- ***L2in***

  The name of the second layer to be XORed.

- ***Lout***

  The name of the output layer for the operation.

Macros

Macros are application extensions that you write to add new functionality to your application. To use macros, you must first add them to the Macros menu. For a complete discussion of creating macros, refer to Chapter 5, "Writing Macros".

Macros create

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a macro.

Usage

Macros create *macroName commandProc guiPlugin*

Arguments

- ***macroName***

  A required argument assigning a name to the macro.

- ***commandProc***

  A required argument specifying the Tcl procedure that performs the macro processing. This procedure can have no dependence on the use of Tk.

- ***guiPlugin***

  A required argument specifying the Tcl/Tk procedure that interfaces to the application.

Macros delete

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Removes the macro from the list of known macros. The macro still appears in the **Macros menu** until the **Macros** menu command is re-invoked.

Usage

Macros delete *macroName*

Arguments

- ***macroName***

  The name of the macro to be deleted.

Macros menu

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Re-creates the **Macros menu** with all your currently defined macros added.

Usage

Macros menu

Macros show

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns information about the macros currently registered with the application.

- When no arguments are specified, the command returns a list of all registered macros.
- When the *macroName* is supplied, the command returns the names of the macro command procedure, and the macro gui plug-n procedure.

Usage

Macros show [*macroName*]

Arguments

- *macroName*

  An optional argument specifying the name of the macro about which you are requesting information.

StringFeature Object

The StringFeature object is a set of polygons in the shape of alphanumeric characters, which you can place in a layout. This differs from the text objects in a layout in that StringFeature objects are geometrical, while text objects are non-geometrical.

- **Use the StringFeature command** to create a StringFeature.
- **Use the StringFeature object method,** $str addToLayout**,** to add the StringFeature to your layout.

You reference a StringFeature object by its handle, which in the case of a StringFeature, is the name you assign to it.

Note
In the reference pages that follows, the variable $str always refers to the handle of an actual StringFeature object.

StringFeature

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Creates a stringfeature.

Returns

The handle of the stringfeature

Usage

StringFeature *name x y w h charsize "string" justify*

Arguments

- ***name***

  Required argument defining the name of the StringFeature you are creating. Every StringFeature must have a unique name, because the name is also used as its handle.

- ***x y***

  Required argument defining the X and Y coordinates of the lower left corner of the bounding box for the StringFeature. When you place the StringFeature in a layout, these coordinates are snapped to the grid defined in the addToLayout method.

- ***w h***

  Required arguments defining the width and height of the bounding box for the StringFeature, in um.

- ***charsize***

  Required argument defining the size of the characters, in um. All characters are designed to fit into a square box, with sides equal to ***charsize***. If the ***charsize*** is too large to fit the entire string into the bounding box, the application adjusts the ***charsize*** down to fit.

- ***string***

  Required argument defining text to be represented as polygons. Must be enclosed in quotes.

- ***justify***

  Required argument defining how the string is justified in the bounding box. Allowed values are:

  - l — left
  - r — right
  - c — center

Example

Assume we have layout0, with TOP cell, and layer 1, we use the StringFeature command with the $str addToLayout command as follows:

```
set strobj [StringFeature abc 0 0 100 100 100 "XYZ string" 1]
$strobj addToLayout layout0 TOP 100 5
#... add to other layout if needed here.
delete object $strobj
```

You will have to either perform a Zoom All or otherwise update the viewer to see the new object.

$str addToLayout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Inserts a stringfeature into an existing layout. Note that this is a StringFeature method, which means the method name is preceded by the handle for the StringFeature to be added to the layout.

Usage

$str addToLayout *layout_handle cellname grid layer*

Arguments

- ***layout_handle***

  The handle of the layout into which you want to insert the StringFeature.

- ***cellname***

  The name of the cell into which the StringFeature will be inserted.

- ***grid***

  The grid to which the character polygons will be snapped, specified in layout dbus.

- ***layer***

  The layer on which the StringFeature is to be added.

cwbLayoutView Object

The cwbLayoutView class holds information related to a view of a particular layout.The cwbLayoutView objects are created for you automatically, hence there are not commands for creating them. The **cwbLayoutView object methods**[1] let you solicit information about what is displayed there.

You reference the cwbLayoutView object by its handle. As with all Tcl and Tk objects, the cwbLayoutView handle also acts as a command for operations related to that object. You can access the cwbLayoutView handle using the $cwb cget -_layoutView command.

In the reference pages that follow, the variable $V always refers to the handle of an actual cwbLayoutView object.

$V cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns the name of the currently viewed cell.

Usage

$V cell

1. With Tcl, an object is referenced by its handle. The handle also acts as a command for operations performed on the object. Thus, the handle for the layout you are working on becomes the name of the commands you use to manipulate the layout data.

$V databaseToScreenUnits

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Translates a pair of x and y coordinates from database units to screen units.

Usage

$V databaseToScreenUnits ***x y***

Arguments

- ***x y***

  The x and y coordinate values expressed in database units that you want to have converted into screen units.

Returns

The x and y coordinates expressed in screen units.

$V depth

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Returns the current view depth.

Usage

$V depth

$V exportView

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Exports the current view as an image file..

Usage

$V exportView *imagetype filename*

Arguments

- ***imagetype***

  The image type to output, which must be one of:

  - JPG
  - PNG
  - GIF
  - BMP
  - XPM
  - PPM
  - TIFF

- ***filename***

  The file name to output the image as. If a file already exists by that name in the destination directory, it is overwritten.

$V micronToScreenUnits

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Translates a pair of x and y coordinates from microns to screen units.

Usage

$V micronToScreenUnits ***x y***

Arguments

- ***x y***

  The x and y coordinate values expressed in microns that you want to have converted into screen units.

Returns

The x and y coordinates expressed in screen units.

$V screenToDatabaseUnits

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Translates a pair of x and y coordinates from screen units to database units.

Usage

$V screenToDatabaseUnits ***x y***

Arguments

- ***x y***

  The x and y coordinate values expressed in screen units that you want to have converted into database units.

Returns

The x and y coordinates expressed in database units.

$V screenToMicrons

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

Description

Translates a pair of x and y coordinates from screen units to microns.

Usage

$V screenToMicrons ***x y***

Arguments

- ***x y***

  The x and y coordinate values expressed in screen units that you want to have converted into microns.

Returns

The x and y coordinates expressed in microns.

Chapter 7
WORKbench Batch Commands

Note

The commands documented in this section function only within Calibre WORKbench.

Command Listing

The modeling commands are described in the sections that follow.

Table 7-1. Modeling Commands

Function	Description
aerial	Simulates the aerial image intensity within an area defined by the specified bounding box.
bestdose	Finds the optimized second exposure dose value.
bestsrcregion	Optimizes source pixels to achieve the best image fidelity for the specified source map.
cdruler	Returns measurement data for each of the sample points in the specified spreadsheet file.
cdsample	Simulates the sites specified in the samfile, then return resulting simulation data.
clipLayout	Writes out the gauges within the specified clip to a layout file.
contourGenerate	Generates aerial/resist contours.
cutline	Simulates the aerial image intensity along a cutline.
cutline2D	Simulates the cutlines for several z-planes within the resist to characterize standing waves in the resist.
dprintiml	Simulates a dense printimage and returns one or more of the following: dense printimage, reference layer, threshold image.
etchtp	Generates a version 10 etch test pattern and associated spreadsheet.
fitmap	Calculates the composite source that best fits the specified source map.
gauge2layout	Writes gauges to a layout.
gaugeGet	Gets data about a row in a gauge object.

Table 7-1. Modeling Commands

Function	Description
gaugeMeasure	Measures gauges on a specified layer.
gaugeRead	Reads a gauge file into a layout.
gaugeSet	Sets data for a specified gauge or downsizes a gauge object.
gaugeWrite	Produces a gauge file from a gauge object.
gmWrite	Writes gauge measurements to a gauge file.
iripple	Tests the interaction radius between features for use in evaluating ripple distance.
kernsym	Reports on model kernel symmetry.
layout2gauge	Transforms wires to gauges.
maitgen	Generates a jobdeck from multiple test pattern inputs.
maskspectrum	Generates a mask spectrum file for the specified layout.
mergemap	Combines multiple source map files into a single averaged source map file.
modelfitness	Runs a simulation using the models defined in the setup file, to produce model fitness statistics.
modelflow	Creates an optimized VT5 model and an optimized optical model.
odanalyze	Analyzes errors induced in the optical model from truncating the optical diameter.
openlib	Loads a GDS file, a setup file, or both into the Calibre WORKbench or LITHOview application.
opticsgen	Creates an optical model.
resistgenv2	Creates a CTR, VTR, or VTR-E resist model.
samlinev2	Characterizes the aerial image in terms of Imax, slope, and other properties.
scalfit	Measures the fitness value of a SEM calibration contour.
scalcontour	Creates a SEM calibration contour.
ss_markup_layout	Draws boxes in a GDS file at lotions indicated by the spreadsheet.
ss2gauge	Produces a Tcl gauge object.
ssclean	Reports or fixes non-centered sample sites.
sss_add	Adds a spreadsheet object to a super spreadsheet object.

Table 7-1. Modeling Commands

Function	Description
sss_cget	Retrieves information about a spreadsheet in a super spreadsheet object.
sss_config	Updates specifications for a spreadsheet inside a super spreadsheet object.
sss_create	Creates a super spreadsheet object.
sss_delete	Deletes a spreadsheet within a super spreadsheet object.
sss_save	Saves a super spreadsheet object.
tepCompile	Generates a custom test pattern based on specifications in a TPF file.
tepCreate	Generates a version 11/12 test pattern and associated spreadsheet.
tepCreate etch	Generates a version 11 etch test pattern and associated spreadsheet.
tepScal	Generates a SEM Calibration test pattern and associated spreadsheet.
tepVerification	Generates a verification test pattern and associated spreadsheet.
tpcreate	Generates a version 10 test pattern and associated spreadsheet.
utp::table2sparse	Converts measurement data from a pitch array into the lookup_file format required for utp::verify.
utp::verify	Generates data comparing simulated CDs to predicted CDs.
vsbIndex	Sets the path and name of an index file for VSB11 chip processing.
vt5gen	Creates a VT5 model.

aerial

Tools Supported: Calibre WORKbench, LITHOview

Description

This command simulates the aerial image intensity within an area defined by the specified bounding box. It calculates the intensity at evenly spaced points defined by an image grid, defined by ***pixsz***.

This command passes its results to the specified grid object. The grid object must exist prior to issuing this command. To inspect the data in text format, you can use the dump method for the grid object to save the data as a text file.

Note

When using hierarchial layouts, the layout must be flattened before it can be used with the **aerial** command.

Requirements

Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load a layout file: layout create
- Load a setup file: openlib
- Create a cwbGrid object: cwbGrid

After issuing this command you can save the simulation to a text file using the $grid dump command.

Usage

cwbDevelop::aerial *gridOb layout pixsz sel_rect hopkinsFlag* [*sigma*]

Arguments

- ***gridOb***

  The name of a cwbGrid itcl object in which to place the aerial image values.
- ***layout***

  The handle for the layout object containing the layout data to simulate.
- ***pixsz***

  The spacing between points in the simulation, in microns.
- ***sel_rect***

  The coordinates of a selection rectangle that defines the portion of the layout, from which to grab the geometry used to simulate the aerial image. The selection rectangle is defined by two coordinate pairs, ***x1 y1 x2 y***2. The first pair defines the lower left corner, the second defines the upper right corner. Coordinates are specified in database units.

You can pass the selection rectangle coordinates in as a variable array, such as the one obtained from $cwb getSelectionRect (`set sel_rect [$cwb getSelectionRect]`) or as four values (-7100 -300 -3700 5600). Note that if you pass the selection rectangle as four values, they must be enclosed in quotes or curly braces "{}" so the Tcl interpreter will recognize them as a single argument.

- **hopkinsFlag**

  A required argument defining the type of model to use for the simulation. Allowed values are:

  0 — uses the SOCS model simulation (faster)

  1 — uses full Hopkins model simulation

- *sigma*

  An optional argument that when present performs two functions:

  a. Instructs the application to convolve with image with a Gaussian filter.

  b. Defines the sigma value for the Gaussian convolution.

Example

```
cwbGrid gridA
cwbDevelop::aerial gridA layoutA 0.01 “0 0 10 10” 0
gridA dump “grid.txt”
```

bestdose

Tools Supported: Calibre WORKbench

Description

This command runs a a uniform search, followed by a Newton optimization to find the best second exposure dose. The optimization objective is the EPE RMS.

- The uniform search has step of 0.1 from *min* to *max* values.
- Newton optimization starts from the best result of the uniform search.

Results are returned in the TCL result string.

Note
You can also run this command in the Model Center GUI using the **Optimization tab** > **Optimize Second Dose** button.

Usage

bestdose -ss *ssfile* [**-l** *layout* | **-m** *gds*] [**-f** *setup*] [**-min** *min*] [**-max** *max*]

Arguments

- **-ss** *ssfile*

  Required spreadsheet file containing CD measurements.
- **-l** *layout*

  Optional layout handle containing a test pattern.
- **-m** *gds*

  Optional GDS file with test pattern.
- **-f** *setup*

  Optional setup file to use with the command.
- **-min** *min*

  Low search bound. Default is **0.5**.
- **-max** *max*

  High search bound. Default is **1.5**.

Return Value

Returns a string with the best second exposure dose and RMS EPE value in the form:

```
-dose bestdose -rms value
```

bestsrcregion

Tools Supported: Calibre WORKbench

Description

Optimizes source pixels to achieve the best image fidelity for the specified source map. Calibre WORKbench writes the resulting source map into the file out. The command log is saved to a file out.log. The bestsrcregion command:

- Ignores the source shape from the setup optical model.
- Defines the initial shape from the source map file specified using -source.
- Conducts simulations only in a scalar regime.
- Uses an algorithm that implements the Abbe method of integration over source.
- Ignores the following modeling options:
  - aberrations
  - image diffusion
  - kernel diffusion
  - vector modeling
  - multiple defocus planes
  - SOCS-related parameters (kernel grid, etc.)
  - reduction factor
  - illumination definition (annular, standard, etc.)
  - beamfocus
  - apodization loss
- Defines the defocus through the “defocus start” value.

Requirements

Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load a layout file: layout create
- Load a setup file: openlib

Usage

bestsrcregion -l ***layout*** **-bbox** {***x1 y1 x2 y2***} **-simpixsz** ***size*** **-source** ***sourcemap***
-cell ***cellname*** [-weight *weightlist*] [-regular *regmethod*] [-gamma *gammavalue*]
[-bgweight *bgvalue*] [-out *outsource*]

- **-l** ***layout***

  A required keyword / value pair specifying a layout handle. Contains geometries which will be simulated.

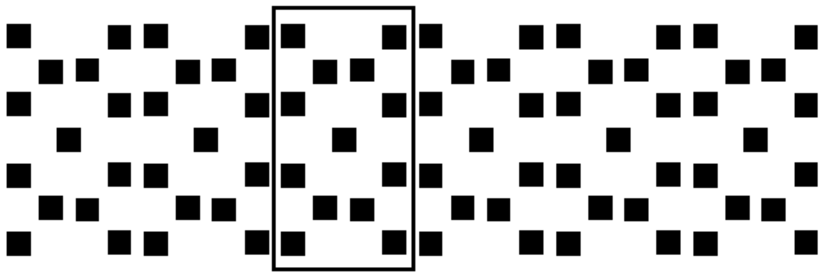
- **-bbox** {***x1 y1 x2 y2***}

  A required keyword and arguments specifying a rectangular region of *layout* from which to grab the geometry used for simulations. The first two values define the lower left corner, the second define the upper right corner. Coordinates are specified in database units,

  Use the **-bbox** bounding box argument to define a rectangle for which image fidelity will be optimized. The bestsrcregion command assumes that this boundary condition is strictly periodic (unlike other WORKbench simulation commands).

  The following figure shows a properly defined simulation *bbox* for a periodic array of the contact holes:

Figure 7-1. Properly Defined Simulation *bbox*



- **-simpixsz** ***size***

  A required keyword specifying a size value for the simulation pixel for GDS, in um. It must be an integer multiple of the optical diameter.

- **-source** ***sourcemap***

  A required keyword specifying an initial source map file. It is used to constrain area in the source that is subjected to optimization. Only those source pixels are optimized that are lit in the *sourcemap*. The source definition from the setup optical model is ignored.

- **-cell** ***cellname***

  A required keyword specifying the name of the cell in the specified layout to be simulated.

- -weight *weightlist*

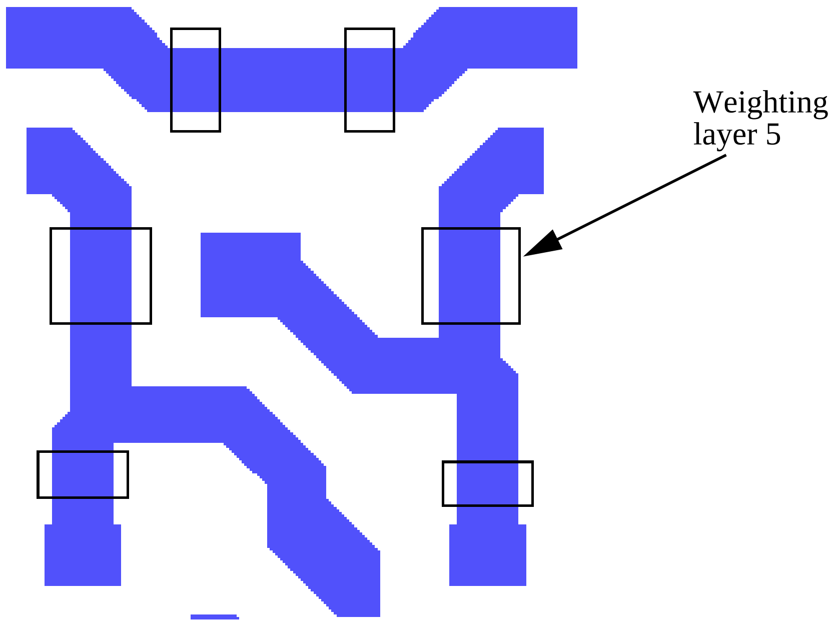
  An optional keyword specifying a list of layers and their weights to accentuate important areas in the layout. The format is {layer1 weight1 [layer2 weight2] ... }

Use the -weight option to optionally define weighting layers. For example, to set a weight of 64 to the regions in layer 5, specify the following:

```
-weight {5 64}  ...
```

Use the -weight option to accentuate gate areas, similar to as shown in Figure 7-2 (which shows a weighting layer of 5 consisting of 6 rectangles over the gates.

Figure 7-2. Weighting to Accentuate Gate Areas



- -regular *regmethod*

  An optional keyword specifying a value for the method to be used for the source smoothing. Can be 0, 1, or 2. 0 means the source points which are different from the original source map are penalized. 1 means that first derivative is used for smoothing. 2 means that second derivative is used for smoothing. The recommended value is 1.

  Use the *regmethod* parameter to specify how to smooth source during optimization.

- -gamma *gammavalue*

  An optional keyword specifying a value for the balancing parameter of the objective function. Can be between 0 and 1.

  - 1 means that no source smoothing is done (the resulting source will be dark).
  - 0 means that the source will be completely smoothed (the resulting source will be bright).

  The recommended value is 0.38.

The objective function definition incorporates the algorithm balancing parameter -gamma as follows:

$$(\text{Objective})^2 = (\text{gamma} \times \text{Imagefidelity})^2 + ((1 - \text{gamma}) \times \text{SourceSmoothness})^2$$

Generally, using smaller values of *gamma* results in brighter sources. If you do not specify the -gamma parameter, the **bestsrcregion** command switches to an exploratory regime and tries the following values: 0.91, 0.83, 0.71, 0.56, 0.38, 0.29, 0.24, 0.14, 0.07, and 0.04. It writes a source map for each of the values to the current directory.

- -bgweight *bgvalue*

  An optional keyword specifying a value for the background weight for the rectangular simulation region *bbox*. Used in the optimization objective. The default is 1.

- -out *outsource*

  An optional keyword specifying a name for the resulting optimized source map. In addition, Calibre WORKbench saves the command log to a file *out.log*.

Note:

The simulation runtime mainly depends on the following:

- The pixel size in the specified sourcemap
- The simulation pixel size
- The size of the area specified in **-bbox**

Example:

```
#Open setup file
openlib -f curv.in
#Create layout from GDS data
set lin [layout create h9_via.gds]
#Run optimizations
bestsrcregion -l $lin -cell "2x2bit" -simpixsz 0.02 -bbox "0 360 900 2140"
-source quarter_standard.smf -regular 1 -weight [list 5 64 7 32] -bgweight
0
```

cdruler

Tools Supported: Calibre WORKbench, LITHOview

Description

Returns measurement data for each of the sample points in the specified spreadsheet file. Note that you will obtain the most meaningful data if the sample points are located close to the sites you are interested in.

Usage

cdruler -ss *spreadsheet* -f *setup* { **-m *gds*** | **-l *handle*** }

Arguments

- **-ss *spreadsheet***

  A required argument / value pair defining the pathname for the spreadsheet file.
- **-f *setup***

  A required argument / value pair defining the pathname for the setup file.
- **-m *gds*** | **-l *handle***

  A required argument / value pair defining the layout to be measured.
  - If the layout data is not loaded into the application use -m to define the pathname for a valid test pattern file.
  - If the layout data is one of the layouts currently loaded into the application. use -l and specify the handle of that layout.

Return Value

The returned string contains a list of records of the following format:

```
size_in_dbu size_in_nm measureType x1 y1 x2 y2 measureInput
```

where:

- size_in_dbu

  The distance between polygon edges, expressed in database units. See measureType for more information.
- size_in_nm

  The distance between polygon edges, expressed in database nm. See measureType for more information.
- measureType

  The Location setting from the spreadsheet file:
  - 0 = measure the width of the polygon at this location

    - 1 = measure spacing to the nearest polygon

- (x1, y1), (x2, y2)

  The points between which the spacing is measured. (x1,y1) is the point on the left or bottom; (x2,y2) is the point on the right or top.

- measuredInput

  The Measurement setting from the spreadsheet file.

cdsample

Tools Supported: Calibre WORKbench

Description

Instructs the application to simulate the sites specified in the samfile, then return resulting simulation data, including the simulated and original CD values. The CD is calculated as a distance between two edges closest to the specified sites.

This batch command provides three output options:

- Sample Site data
- CD data
- EPE data

Refer to section entitled "Output" for a complete description of each.

Usage

cdsample -ss *spreadsheet* -f *setup* {**-m *gds*** | **-l *handle***} [-o *output*] [-oo *output2*][-allowna] [-e 0|1]

Arguments

- **-ss *spreadsheet***

  A required keyword/argument pair supplying the pathname for the sample points spreadsheet file.

- **-f *setup***

  A required keyword/argument pair supplying the pathname for the setup file.

- -e { 0 | 1 }

  An optional argument specifying behavior when errors occur:

  - 0 — Lets the run complete, appending zeroes to the result line.
  - 1 — Stops at any error (the default).

- **-m *gds*** | **-l *handle***

  A required keyword/argument pair supplying either the pathname for the gds file or a layout object handle. You must specify one or the other and cannot use both.

Note

When specifying a layout object handle, the supplied layout must be a flat database; the **-l *handle*** option does not work with databases containing any hierarchy layers. Use the layout copy command to copy a hierarchical layout to a flat layout for use with this command.

- -o *output*

  An optional keyword/value pair specifying the name of an output file to use for storing CD data. All simulated parameters in this output file are AVERAGED between the two sites used to make the CD. Therefore care should be used if the structures are non-symmetrical, as the simulated parameters may not make sense. Refer to section entitled “CD Output” for a complete description of this file.

- -oo output2

  An optional keyword/value pair defining the name an output file used for storing EPE data. Refer to section entitled “EPE Output” for a complete description of this file.

- -allowna

  An optional keyword that allows spreadsheets to be added with “NA” string values for their measurement values instead of aborting the command (the default option).

Output

This command allows you to choose between three types of output:

The return string

The cdsample command returns a string containing a set of ten values per site. A colon “:” separates the data for one site from the next. The ten values, in order, represent the following information:

- CD — simulated CD on wafer, in nm (this is the most important information)
- CDOrig — original CD on mask, in nm
- MeasType — 0 or 1, space or width CD, in nm
- x1 — coordinate of the first edge, in dbu
- y1 — coordinate of the first edge, in dbu
- x2 — coordinate of the second edge, in dbu
- y2 — coordinate of the second edge, in dbu
- epe1 — first edge EPE, in nm
- epe2 — second edge EPE, in nm
- MeasCD — measured experimental CD at the site, in nm

CD Output

The -o output switch creates a file using the Sample Data File format and containing the aerial image and CD data for each of the sampling site in a test pattern spreadsheet.

Each row in the output file contains the following data:

- column 1 = imax - using definition of nominal model

- column 2 = slope - using definition of nominal model
- column 3 = islope - using definition of nominal model
- column 4 = imin - using definition of nominal model
- column 5 = factor - using definition of nominal model
- [column 6 = d1] - densities come from nominal model spec
- [column 7 = d2]

  [...]
- column M-4 = empthresh, the empirical printing threshold
- column M-3 = modelthresh, the model predicted threshold
- column M-2 = empepe, the empirical EPE
- column M-1 = modelepe, the model predicted EPE (thresh + bias)
- column M = EPEv, EPE from the variable threshold (no bias)

EPE Output

The -oo output switch uses the same format as the -o option (the Sample Data File format). However, the data represents edge placement errors rather than CDs. It contains the aerial image and EPE data for the two edges of a sampling site.

Since a CD is formed by two edges, there are as twice as many rows as there are sample sites. The rows in the first half of the file represent the edges on one side of the feature, and each contains information about one simulated EPE of the CD. The rows in the second half of the file represent the edges on the other side of the CD measurements, and each contains information on the simulated EPE opposing an edge in first half of the file.

The first and N+1 rows represent opposing EPEs, as do the second and the N+2 row, etc. (where N is the number of CDs). Thus, when there are N lines in the spreadsheet used as input to the command, this file contains 2*N lines.

Note that the empirical printing threshold (column M-4) is the "single" empirical threshold that matches the simulated CD. Thus the empirical threshold is always the same between the first and N+1 row, the second and the N+2 row, etc. (where N is the number of CDs)

Example

The following example shows the command in its basic form (no -o or -oo output files are written to) issued at the Tcl prompt. Note that the return string is saved as “a”, then echoed to a txt file for examination.

```
% set a [cdsample -s samdata -f ./example/test.in -m ./example/test.gds]
:
: <output>
:
% echo $a > cdsample_data.txt
```

The following is an example of the output for a single site:

```
209.134059 201.250000 0 46862 3992 46908 2871 3.943565 3.940494 200
```

These values correspond to the following items:

Table 7-2. cdsample Command Output

Output Item	Value
Simulated CD(Printimage CD)	209.134059
Mask CD	201.250000
Other(0 or 1)	0
x1 y1 x2 y2	46862 3992 46908 2871
EPE(x1,y1)	3.943565
EPE(x2,y2)	3.940494
Measured CD	200

clipLayout

Tools supported: Calibre WORKbench

Description

Flattens/clips the given layout, filters out the parts not in vicinity (inside the search rectangle specified; this is also known as ‘touch mode’) of any gauge, and writes the result to a Calibre WORKbench layout. Note that you must have previously created the destination layout.

The clip “search rectangles” used are temporary boxes drawn around the gauges. For a given gauge (specified by two corners x1,y1 and x2,y2 as the coordinates of the left bottom and right top corners), a search rectangle is created using the following formula:

```
(min (x1,x2) - extra, min (y1,y2) - extra)
```

and

```
(max (x1,x2) + extra, max (y1,y2) + extra)
```

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section “Gauge Object Command Relationships” in Chapter 8, “WORKbench File Formats.”

Usage

clipLayout -g *gauge_data* [-l *wblayout* | -m *layoutfile*] **-extra *extra* -o *output_wblayout***

Arguments

- **-g *gauge_data***

  A required argument specifying a gaugedata Tcl object (produced with ss2gauge or gaugeRead)

- -l *wblayout*

  An optional argument specifying an input layout object previously loaded into Calibre WORKbench.

- -m *layoutfile*

  An optional argument specifying an input layout in one of the supported formats (such as GDS or OASIS)

- **-extra *extra***

  A required argument specifying a value in microns to enlarge the search rectangle by.

- **-o *output_wblayout***

  A required argument specifying a WORKbench output layout name.

Example

```
#create a gauge object
set gobj [gaugeRead -f “gauge.gg”]
#load source layout
set l_src [layout create “tp.oas”]
# create empty layout to use as output
set l_clip [layout create]
$l_clip create cell “TOP”
#clip
clipLayout -g gobj -l $l_src -extra 10- -o $l_clip
#output clipped layout to file
$l_clip oasis_out “tp_clipped.oas”
```

contourGenerate

Tools supported: Calibre WORKbench

Description

Generates aerial/resist contours, according to the models provided in the setup file. The contours are placed on the specified layer.

Usage

contourGenerate -l *wblayout* -outlayer *layer* -f *setupfile* -filter *l.d* [-compact]

Arguments

- **-l *wblayout***

  A required argument specifying the input layout name.

- **-outlayer *layer***

  A required argument specifying a layer number where Calibre LITHO places the contours.

- **-f *setupfile***

  A required argument specifying the Calibre LITHO setup filename to read required process parameters from. Additionally, the setup file must have a sse LITHO_DENSE_PRINT command specified within it in order for contourGenerate to work correctly.

- **-filter *l.d*** [-compact]

  An optional argument specifying that Calibre WORKbench should use the indicated layer as a filter layer for the operation. The *l.d* specified as a filter can be defined in the setup file as any layer type. If it is not defined in the setup file, it is automatically added as a "hidden" layer.

  The [-compact] option speeds up contour generation instructs Calibre WORKbench to use the gauge objects on the filter layer as clip selection boxes (only objects that contain features crossed by gauges will be considered).

  You use the gauge2layout command to create the gauges.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Example

Generates contours on the entire layout:

```
contourGenerate -l layout0 -outlayer 10 -f litho.in
```

Generates contours using the filter layer by gauges:

```
set g_obj [gaugeRead -f "gauge.gg"]
```

```
    gauge2layout -g g_obj -l layout0 -outlayer 8 -boxes 400
    contourGenerate -l layout0 -outlayer 10 -f litho.in -filter 8 -compact
```

cutline

Tools Supported: Calibre WORKbench, LITHOview

Description

This command simulates the aerial image intensity along a cutline. It calculates the intensity at the two endpoints and at evenly spaced intervals along the cutline. The output is a Tcl result string that is a list of {x y value} triplets, representing (x,y) coordinates where the intensity was simulated, and the value of the aerial image at (x,y).

Requirements

Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load a layout file: layout create
- Load a setup file: openlib

Usage

cwbDevelop::cutline *x1 y1 x2 y2 layout pixsz* *hopkinsFlag*

Arguments

- ***x1 y1 x2 y2***

  The coordinates of the two endpoints of the cutline to simulate, in database units.

- ***layout***

  The layout object containing the layout data to simulate.

- ***pixsz***

  The spacing between sampling points on the cutline, specified in microns.

- *hopkinsFlag*

  Optional argument specifying whether to use SOCS approximation or the Hopkins with Fourier series periodization of the mask.

  - 0 for SOCS (This is the default.)
  - 1 for Hopkins

Example

```
cwbDevelop::cutline “0 0 100 0” layoutA 0.01
--> {{0 0 .32} {0 10 .342} ...}
```

cutline2D

Tools Supported: Calibre WORKbench, LITHOview

Description

This command simulates the aerial image intensity along chosen points on the cutline. It computes the cutlines for several z-planes within the resist. This can be plotted as I(x,z) to allow you to view standing waves in the resist.

Requirements

Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load a layout file: layout create
- Load a setup file: openlib
- Create a cwbGrid object: cwbGrid

After issuing this command you can save the simulation to a text file using the $grid dump command.

Usage

cwbDevelop::cutline2D *gridOb edge inLayout pixsz imagePlanes*

Arguments

- ***gridOb***

  The name of a cwbIntensity itcl object for the output aerial image values.
- ***edge***

  The cutline edge to simulate, as expressed as {x1 y1 x2 y2} in database units.
- ***inLayout***

  An existing layout object from which to obtain the geometry.
- ***pixsz***

  The spacing between points in the simulated cutline, in microns.
- ***imagePlanes***

  The planes to simulate the aerial image in the format {begin end num}.

Example

```
% cwbGrid gridA
% cwbDevelop::cutline2D gridA “0 0 100 0” layoutA 0.01 “0 .3 40”
--> {}
% gridA dump “data.txt”
```

dprintiml

Tools Supported: Calibre WORKbench, LITHOview

Description

Performs a dense printimage simulation using a VT5 model and returns up to three separate images:

- **Dense Printimage** — the PRINTimage generated using the entire model, which uses a threshold polynomial and/or a bias polynomial.
- **Reference Image** — a contour of the aerial image at the (constant) reference threshold. This represents the optical portion of the model.
- **Threshold Image** — the PRINTimage generated using the threshold polynomial alone. This represents the resist portion of the model.

Requirements

Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load an input layout file: layout create
- Load a setup file: openlib

Usage

dprintiml -in *inlayout* -bbox *x1 y1 x2 y2* -out *outlayout* -cell *outcell* -layer *layer* -simpixsz *simpixsz* [-morphsize *msize* -reflayer *reflayer* -thlayer *thlayer*]

Arguments

- **-in *inlayout***

  A layout handle. Contains geometries which will be simulated.

- -bbox ***x1 y1 x2 y2***

  A rectangular region of the layout from which to grab the geometry used for simulations. The first two values define the lower left corner, the second define the upper right corner. Coordinates are specified in database units.

- **-out *outlayout***

  A layout handle. Results of simulations will be written to this layout as layers. This layout must exist when dprintiml is called. It can be the same layout handle as *inlayout*.

- **-cell *outcell***

  The name of the cell in *outlayout* to output results.

- **-layer *layer***

  The layer number from *outlayout* to output results. May or may not exist prior to simulations.

- **-simpixsz *simpixsz***

  The size of the simulation pixel, in um. It must be an integer multiple of the optical diameter. It must also be a power of 2 multiple of hoodpix.

- -morphsize *morphsize*

  An optional size of over/under and under/over cleanup operations, in nm. Default value is 0 (no cleanup).

- -reflayer *reflayer*

  An optional layer number in outlayout to output the VT5 reference layer. May or may not exist prior to simulations. If 0, then reference layer will not be outputted. Default value is 0.

- -thlayer *thlayer*

  An optional layer number in outlayout to output the VT5 threshold layer. May or may not exist prior to simulations. If 0, then threshold layer will not be outputted. Default value is 0.

etchtp

Tools Supported: Calibre WORKbench

Description

Generates a version 10 etch test pattern and associated spreadsheet file.

Usage

etchtp *target filename*

Arguments

- ***target***

  The target CD for test pattern structures.

- ***filename***

  The name of the output GDS file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The application also uses this string to generate the name of the spreadsheet file, by stripping away the file extension and appending the extension *.ss*.

fitmap

Tools Supported: Calibre WORKbench

Description

This command calculates the parameters needed to represent the specified source map as a composite source. To calculate the source map, the command first performs a full-factorial search for the main (non-smoothing) parameters with a step value of 0.1 to find a preliminary solution. It then switches to Quasi-Newton[1] optimization for as many as seven parameters of the composite source: sigma out, sigma in, delta sigma out, delta sigma in, illumination angle, delta illumination angle, and weight. The optimization objective F is the norm-1 difference between input source map S and composite source map C:

$$F = \iint |S(x, y) - C(x, y)|\,dx\,dy$$

Return Value

fitmap returns a string containing the parameters of the composite source that best approximate the input source map.

For annular, quasar, standard, and cross-quad types, this string is in the form:

```
-composite {weight w sigma s1 sigma_in s2 illumangle ia dsigma ds1
dsigma_in ds2 dillumangle dia} -fit objective_value
```

The weight parameter is optimized for all optimizations. You control which other parameters to optimize through -profile profiletype and -type comptype.

For quad and dipole types, this string is in the form:

```
-composite { weight w sigma_cntr s1 radius r rotangle rotangle
dradius dr} -fit objective_value
```

The value of rotangle is not fixed, and can be set in the **-rotangle** command argument.

For a complete description of how these parameters are used to define a source map, refer to Syntax for Specifying a Composite Source.

Usage

fitmap -in *filein* -type *comptype* -profile *profiletype* [-rotangle *angle*] [-out *fileout*]
[-limit *limitflag*]

Arguments

- **-in *filein***

  A required argument / value pair defining the source map file to be fit.

1. This is implemented as in P.E.Gill, W. Murray, and M.H.Wright, “Practical Optimization”, Academic Press, San Diego, 2004.

- **-type *comptype***

  A required argument / value pair defining the type of the composite source that is used to fit the input map. Allowed values are:

  - annular — when specified, optimizes sigma and sigma_in.
  - quasar — when specified, optimizes sigma, sigma_in and illumangle.
  - standard — when specified, optimizes only sigma.
  - dipole — when specified, optimizes cluster radius, spot radius, and angle.
  - quad — when specified, optimizes cluster radius, spot radius, and angle.
  - cross_quad — when specified, optimizes sigma and sigma_in.

- **-profile *profiletype***

  A required argument / value pair defining the *profiletype* for the source. Allowed values are:

  - erf — when specified, optimizes the “d” parameters as well as the base parameters. The “d” parameters (dsigma, dsigma_in, dillumangle) define the transition length. They are also referred to as the user-defined delta ranges, in which the intensity rises from 0.5% to 99.5%.
  - sharp — when specified, optimizes only the base parameters as needed for the -type.

- -rotangle *angle*

  An optional argument specifying an *angle* in degrees. This is used as the rotation angle for the **dipole** and **quad** types, calculated counter-clockwise from x-axis. For example, specifying 90 represents a vertical dipole. Rotation angle is fixed and is not optimized. Default is 0.

- -out *fileout*

  An optional argument / value pair instructing the application to write the sourcemap that results from fitting to the specified file. When not specified, the application does not save the sourcemap.

- -limit *limitflag*

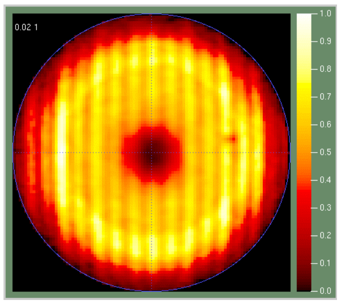
  An optional argument / value pair defining the type of optimization to perform: constrained or unconstrained optimization. Allowed values are:

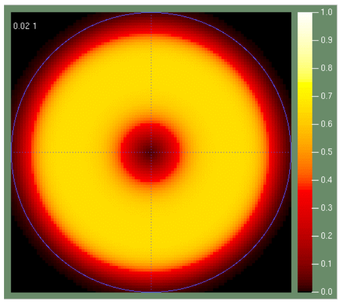
  - 1 — Constrained optimization.
    - Limits sigma, dsigma, sigma_in, and dsigma_in,to values between 0 and 1.
    - Limits illumangle and dillumangle to values between 0 and 90 degrees.
  - 0 — Unconstrained optimization.

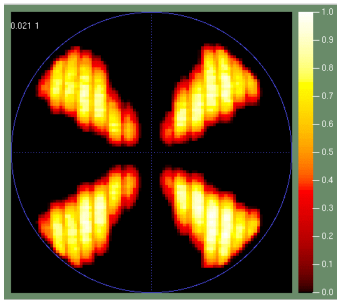
  The default is **1**.

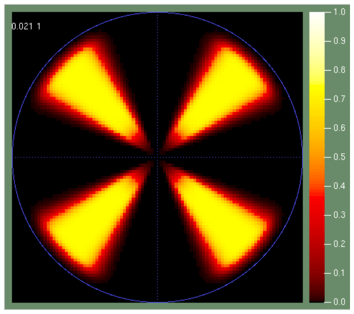
Examples

Original **FItted**









gauge2layout

Tools supported: Calibre WORKbench

Description

Writes a physical representation of gauges in a gauge data object to a layout object.

Tip: For more information on gauge objects and the relationship between the various gauge commands,, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gauge2layout -g *object* -l *wblayout* **-outlayer *layer*** [-ssg][-boxes *size*]

Arguments

- **-g object**

  A required argument specifying an input gauge data Tcl object (produced with ss2gauge or gaugeRead).

- -l *wblayout*

  An optional argument specifying a previously-loaded Calibre WORKbench layout object to read the measurements from.

- **-outlayer *layer***

  A required argument specifying the layer number where to put the gauge output.

- -ssg

  An optional argument specifying that gauge2layout will output subgauges as well as main gauges (ShowSubGauges).

- -boxes *size*

  An optional argument. If specified, this option causes the gauges to be output as boxes instead of wires. The boxes are taken to be the bounding box of the wire and are sized up by the *size* value in database units (dbu).

Example

```
gauge2layout -g aobj -l layout0 -outlayer 3 -ssg
```

gaugeGet

Tools supported: Calibre WORKbench

Description

Gets the specified data associated with the gauge in a gauge object.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gaugeGet -g gauge_object [-size] | [-row *index* [-comment]] | [-cds] | [-cdse] | [-enabled]

Arguments

- **-g *gauge_object***

  A required argument specifying a gauge data object.

- -size

  An optional argument that returns the total number of gauges in the input gaugedata object.

- -row *index*

  An optional argument, requesting the gauge at the specified *index*. Valid values for *index* are integer numbers between 0 and the total number of gauges in the input gaugedata object.

- -comment

  An optional argument that returns the comment associated with the specified row. Must be used with the -row argument.

- -cds

  An optional argument that returns the measured and simulated resist CD data of the specified row. If -row is not specified, this argument returns the measured and simulated resist CD data for all gauges.

- -cdse

  returns measured and simulated etch CD data of the specified row; if row is not specified, returns measured and simulated etch CD data for all gauges (optional)

- -enabled
  returns the enable/disable flag of the specified row; if row is not specified, returns the total number of enabled rows (optional)

Example

```
gaugeGet -g gobj ; # get all gaugedata
gaugeGet -g gobj -size ; # get total number of gauges
gaugeGet -g gobj -enabled ; # get number of enabled gauges
gaugeGet -g gobj -cds ; # get resist CD data for all gauges
gaugeGet -g gobj -row 2 ; # get gaugedata for row 2
gaugeGet -g gobj -row 127 -cds ; # get resist CD data for row 127
gaugeGet -g gobj -row 0 -comment ; # get comment for row 0
gaugeGet -g gobj -row 0 -cdse ; # get etch data for row 0
gaugeGet -g gobj -row 2 -enabled ; # get enabled flag for row 2
```

gaugeMeasure

Tools supported: Calibre WORKbench

Description

Collects measurements based on the gauge file and the layout, creating a gauge measurement object. This is a transitional state to producing a gauge measurement file (with the gmWrite command).

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gaugeMeasure -g *object* [-l *wblayout* | -m *layoutfile*] **-layer *layerno***

Arguments

- **-g *object***

  A required argument specifying a gaugedata Tcl object (produced with ss2gauge or gaugeRead).

One of the -l or -m arguments must be specified. They are mutually exclusive with each other.

- -l *wblayout*

  An optional argument specifying a previously-loaded Calibre WORKbench layout object to read the measurements from.

- -m *layoutfile*

  An optional argument specifying a GDS, OASIS, or other design file name.

- **-layer *layerno***

  A required argument specifying the layer number where the gauge measurements are taken.

Example

```
set aobj [gaugeRead “/tmp/gauge.out”]
set bobj [gaugeMeasure -g aobj -m “sram.gds” -layer 0]
```

gaugeRead

Tools supported: Calibre WORKbench

Description

Reads a gauge object from a gauge file.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gaugeRead -f *filename*

Arguments

- **-f *filename***

  A required argument specifying the input gauge data file name.

Example

```
set aobj [gaugeRead -f "/tmp/gauge.out"]
```

gaugeSet

Tools supported: Calibre WORKbench

Description

Resizes the specified gauge object or resets object data asscociated with one of the gauges.

- For resize operations, all gauges with indices greater than or equal to the specified size (integer number) will be deleted.
- For reset operations, specify the gauge to be changed by its index.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gaugeSet –g *gauge_object* {–size *new_size* | –row *index* [–data *string* –comment *string*]}

Arguments

- **–g *gauge_object***

  Is the input gaugedata object (required).

- –size *new_size*

  Selects resizing mode. Resizes the input gaugedata object to only have as many gauges as the specified size. Deletes any gauges that have index numbers greater than or equal to new_size (which cannot be greater than the size of the input gaugedata object).

- –row *index*

  Selects gauge editing mode. The gauge data at the specified index (row number in the gauge file) is modified, if it exists. It must be followed by either a -data argument, -comment argument, or both.

- –data *string*

  Is an input gauge string containing valid gaugedata format.

- –comment *string*

  Is an input string that is added to the specified gauge entry as a comment.

Example

```
# resize gauge object
gaugeSet aobj –size 274
# reset data for gauge No 46
gaugeSet aobj –row 46 –data "1 iso_cd 1 1 20750 10500 21250 10500 0 100 100
83.9 -1 -1 -1 -1 1 0 0 0" –comment "this gauge was modified"
```

gaugeWrite

Tools supported: Calibre WORKbench

Description

Produces a gauge file out of a gauge object. The output gauge file format will have a syntax similar to that of a spreadsheet file.

The output gauge file format is the following:

```
flag structure_name row col x1 y1 x2 y2 loc drawn other meas simulated \
precision weight mode count dist [# comment]
```

Tip: For more information on gauge objects (including the file format) and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gaugeWrite -g *object* -o *filename*

Arguments

- **-g *object***

  A required gauge data object (produced with the ss2gauge or gaugeRead commands).

- **-o *filename***

  A required argument specifying the output file name for the gauge file.

Example

```
gaugeWrite -g aobj -o "/tmp/gauge.out"
```

Notes

You must explicitly pass the object itself to gaugeWrite; the function does not accept expanded ($-type) variables. (For example, use aobj; do not use $aobj.)

gmWrite

Tools supported: Calibre WORKbench

Description

Writes a gauge_measurement object (produced with gaugeMeasure) to the specified filename. The output gauge measurement file will have lines with the following syntax:

measurement location errorcode

where:

measurement is the measured value

location is 0 for CD, 1 for space measurement

errorcode is an integer number which specifies the type of error that occurred:

- 0 — no error
- 1 — no edges intersecting this gauge
- 2 — too few edges in this gauge
- 3 — too many intersections intersect this gauge
- 4 — ambiguous measurement (Calibre WORKbench cannot ascertain whether a space or CD was measured; this can be the case of polygons included in each other or a corrupted database)
- 5 — nonmatching measurement (CD expected, space measured or vice versa)
- 6 — multiple errors (for multiple measurements per gauge)

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

gmWrite -gm *object* -o *filename*

Arguments

- **-gm *object***

  Specifies a gaugewrite object, produced using the gaugeMeasure command.

- **-o *filename***

  Specifies an output file name for the new gauge_measurement object.

Example

```
set aobj [ss2gauge -ss sram.ss -m "sram.gds" -layer 0 -extra 10]
```

```
gaugeWrite -g aobj -o “/tmp/gauge.out”
# use an external editor to modify gauge.out, if you want


set aobj [gaugeRead -f “/tmp/gauge.out”]
set bobj [gaugeMeasure -g aobj -m “sram.gds” -layer 0]
gmWrite -gm bobj -o “/tmp/gaugemeasurement.out”
```

iripple

Tools Supported: Calibre WORKbench

Description

This command tests the interaction radius ripple on a given setup file. When running this automated test, the application performs he following steps:

1. Read the given setup file.
2. Create a layout containing a set of long horizontal lines. These lines will be on the layer that is defined to be the opc layer in the setup file.
3. Perform OPC and write the results to an output layer, with the layer number:

   ```
   (opc layer number) + -opcbump
   ```

4. Measure the widths of the lines on the OPC output layer and return the list. Width measurements are taken along a vertical line crossing the middle of the lines.

This command returns an ordered list of integers representing the widths of lines on the OPC output layer. The first value represents the first line.

Usage

iripple -f *setup* -width *width* -space *space* [-padwidth *width*]
[-numlines *number*] [-length *length*] [-lout *filename*] [-opcbump *number*]

Arguments

- **-f *setup***

  A required argument / value pair defining the pathname for the setup file being tested.

- **-width *width***

  A required argument / value pair defining the line width, in um.

- **-space *space***

  A required argument / value pair defining the space between lines, in um.

- -padwidth *width*

  An optional argument / value pair defining a unique width for the first line, in um. If not specified or if set to 0, all lines have the same width as defined by -width.

- -numlines *number*

  An optional argument / value pair defining the number of lines to be created, default is 50.

- -length *length*

  An optional argument / value pair defining the length of lines, in um, default 10 um.

- -lout *filename*

  An optional argument / value pair instructing the application to write the test layout to an OASIS formatted file with the specified pathname. If not specified, the test layout is destroyed after processing.

- -opcbump *number*

  An optional argument / value pair defining the layer bump value to use in calculating to put the opc output. The default is 100.

Returns

Example

```
iripple -f setup/nvm.in \
        -space 0.09 \
        -width 0.06 \
        -lout output.oas

Returns -> 201 114 107 122 161 119 127 134 134 132 132 ... 132
```

kernsym

Tools Supported: Calibre WORKbench

Description

Reports on the kernel symmetry of the current optical model. This command returns a list of ordered pairs in which the first value is the kernel number and the second value is either 0 or 1.

- 0 = the kernel is asymmetric and can introduce symmetry problems.
- 1 = the kernel is symmetric.

A model is more accurate if the highest numbered kernel is symmetric.

Usage

kernsym [-f *setup file*] [-num *knumber*]

Arguments

- -f *setup file*

  An argument/value pair that instructs the Calibre WORKbench application to load *setupfile* and the associated optical model. When used, the optical model specified in *setupfile* becomes the current model, and the results report on the kernels for that model.

  This argument is only optional if an optical model has already been loaded into the application; otherwise, it is required.

- -num *knumber*

  An optional argument/value pair defining the number of kernels to test. The default is the number of kernels in the SOCS optical model being evaluated.

Example 1

```
kernsym -num 16

1 1  2 1  3 1  4 0  5 1  6 1  7 0  8 1  9 1  10 1  11 0  12 1  13 1  14 0
15 1  16 1
```

Here, “bad” kernel #'s are 4, 7, 11, and 14, which are not symmetric.

Example 2

```
array set ans [kernsym -f generic.in -num 19]
if {  $ans(10) == 1 } {
   puts “10 is a good number of kernels”
 }
if { $ans(11) == 0 } {
   puts “Avoid 11 kernels”
 }
# Find first good kernel number >= k
set k 9
for { set i $k } { $i < 20 } { incr i } {
  if { $ans($i) == 1 } {
     break;
  }
}
puts “First good kernel number >= $k is $i”
```

layout2gauge

Tools supported: Calibre WORKbench

Description

Reads wires from the layout and transforms them to gauges. The wires should be drawn as paths on the layer specified in the **-layer** argument.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

layout2gauge -l *wblayout* -layer *layer* [-ssg]

Arguments

- **-l *wblayout***

  A required argument, specifying an input layout name.

- **-layer *layer***

  A required argument, specifying the layer to get the gauges from.

- -ssg

  An optional switch, instructing Calibre WORKbench to try to solve for subgauges. The command reads up to 2*n+1 subgauges if this option is specified, and sets the gauge object's count value to the number of gauges found. If -ssg is not specified, it only reads the one central gauge and sets its count value to 0.

Example

```
set bobj [layout2gauge -l layout0 -layer 3]
```

maitgen

Tools supported: Calibre WORKbench

Description

The maitgen (Metrology Automation Interface Tool Generator) command takes multiple GDS test pattern files, the corresponding spreadsheet files, and the coordinates of the lower left corner of each test pattern. It processes the instructions given to orient each spreadsheet, and outputs a job deck for the requested metrology tool.

Essentially, the job deck is a large GDS file that combines all input GDS files with appropriate coordinate shifts. You pass this GDS file to the metrology tool along with the output DBM file. The layout and combined spreadsheet can be directly input into the model building procedure.

maitgen generates GDS output containing all the test patterns in their new placement according to the *coord-lists*. Each test pattern in the *gds-list* is inserted as a cell named with the same name as its top cell (suffixed with an integer, incremented as needed) and the output GDS file has a top cell called TOPCELL. The user specifies the output path.

Note

Calibre WORKbench also includes a GUI tool that replicates the actions in this command: see the section Generating Metrology Tool Jobdecks in the *Calibre WORKbench User's Manual*.

Limitations

- Make sure that all the input test pattern layouts have the same grid size.
- Only Mentor Graphics test patterns are allowed, and they must be complete test patterns.

Usage

maitgen -vendor {amat | hitachi} **-ss *ss-list* -gds *gds-list* -place *coord-list* -deck *metrology-job* -dbu *dbu-in-um* -mask *Mask-Name* -gdsout *combined-gds*** [-ssout *combined-ss*]

Arguments

- **-vendor**

  A required argument specifying the metrology tool vendor:

  - amat for the Applied Materials metrology tool
  - hitachi for the Hitachi metrology tool

- **-ss *ss-list***

  A required list, separated by spaces, of the spreadsheet file names that were generated by the spreadsheet generation commands, The same spreadsheet file may appear multiple times in the list.

- **-gds *gds-list***

  A required list, separated by spaces, of the GDS file names corresponding to the test pattern(s) that were generated by the tpgen command. The same gds file name may appear multiple times in the gds-list.

- **-place *coord-list***

  A required list, enclosed in curly braces (**{ }**), of one or more sets (each set of four is enclosed in curly braces as well) of absolute coordinates (in database units) for the placement of the lower left corner of the test patterns, plus two integer numbers that specify test pattern orientation. It uses the following order:

  ```
  x-shift y-shift mirror rotation
  ```

  - The first two values specify the X and Y shift values in database units.
  - The third value specifies the ***mirror***ing:
    - 0-no mirroring
    - 1-vertical mirror around x-axis
    - 2-horizontal mirror around y-axis
  - The fourth value number specifies the ***rotation***:
    - 0 -no rotation
    - 1 - 90 degree counterclockwise
    - 2 -180 degree rotation
    - 3 - 90 degrees clockwise (270 counterclockwise).

Note

ss-list, gds-list, and coord-list must have the same number of elements, so that 1-to-1 correspondence is maintained.

- **-deck *metrology-job***

  A required name for the metrology job deck output file. The output is dependent on the setting for the **-vendor** argument, and is in the form of an XML file for AMAT, and a spreadsheet (.ss) file for Hitachi.

- **-dbu dbu**

  A required value for the number of database units in a micron. This argument is used for converting coordinates to microns.

- **-mask Mask-name**

  A required argument specifying the mask name used in the AMAT XML file output. This argument must be supplied even if you are using the Hitachi file output.

- **-gdsout *combined-gds***

  A required argument, specifying an output GDS file Calibre creates with all the combined test patterns.

- -ssout *combined-ss*

  An optional argument, specifying an output spreadsheet (.ss) file with all measurements combined. This is essentially a concatenated ss file that you can input into the model building procedures after the CDs column is filled with the measured CDs.

Example 1

Figure 7-3 shows how a test pattern A is placed into three different positions (A1, A2, and A3):

- For placement 1, the coordinate transformation is a simple translation. The following values are entered into the coord-list:

  ```
  112300 4500 0 0
  ```

- Placement 2 is a 90 degree rotation:

  ```
  312300 6700 0 1
  ```

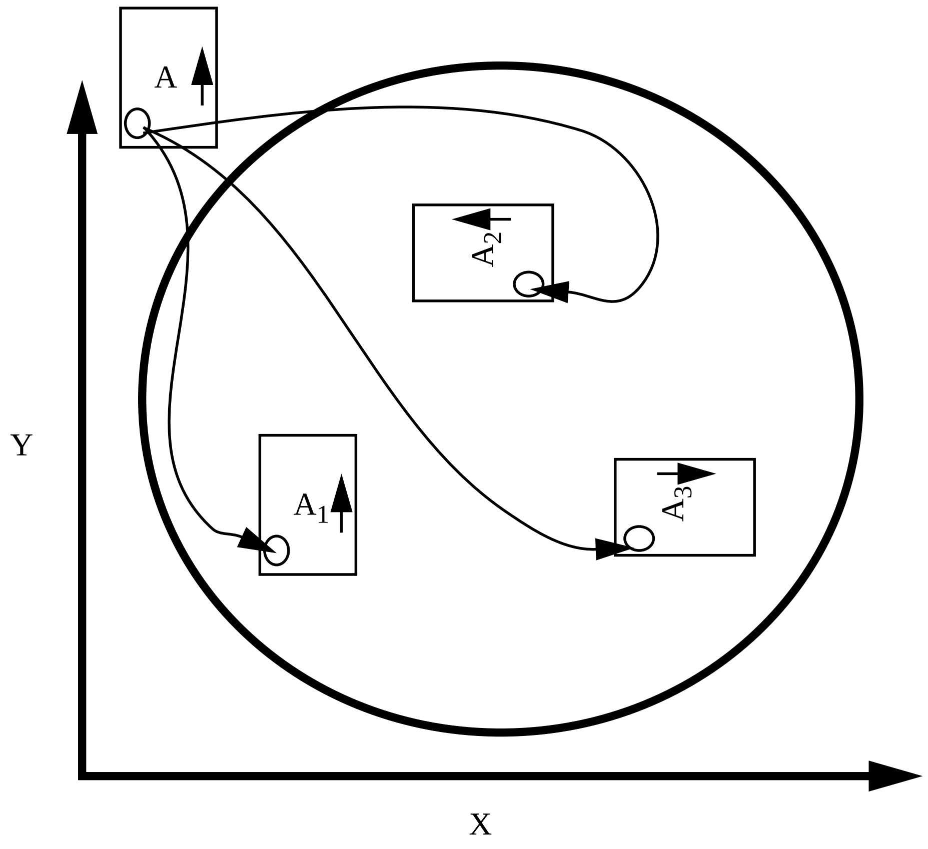
- Placement 3 has mirroring around x=0 and a 90 degree clockwise rotation:

  ```
  412500 4600 1 3
  ```

When invoking maitgen, you combine all three patterns as shown in the following *coord-list*:

```
maitgen -place {{112300 4500 0 0}{312300 6700 0 1}{412500 4600 1 3}} ...
```

Figure 7-3. maitgen Placement Example



Example 2

If you specify the GDS list as follows:

```
maitgen -gds {standard.gds standard1.gds} ...
```

and both layouts have their top cell named "TOP," then the output GDS file contains a top cell called TOPCELL, which contains two cells, TOP0 and TOP1 for standard.gds, and standard1.gds, respectively.

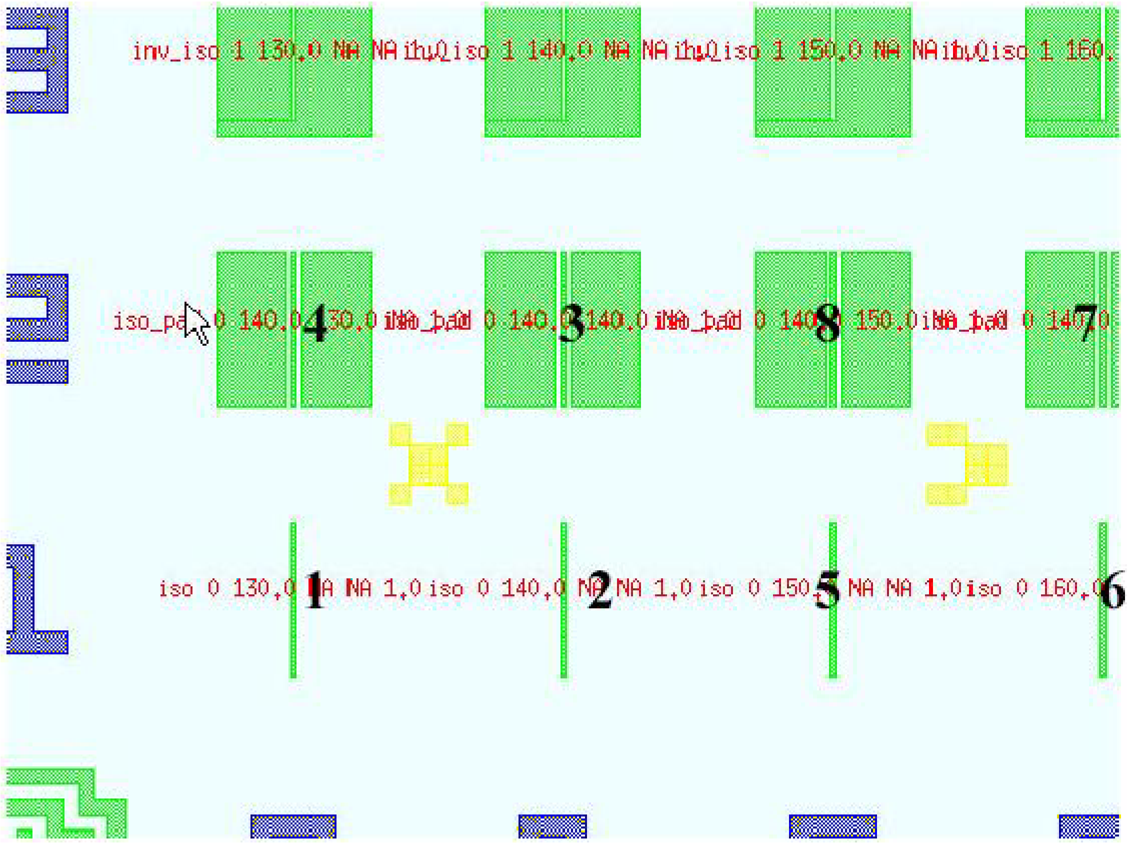
Notes on Hitachi Metrology Deck Generation

Hitachi metrology deck generation requires you to have the Hitachi CD SEM and the Hitachi Design Gauge System.

The layout test pattern is arranged in rows and columns. The arrangement of rows in the Calibre spreadsheet file is such that the first row in the spreadsheet points to the structure in the first row and the first column of the layout, the second row in the spreadsheet points to the structure in the first row and second column in the layout, and so on until the first row in the layout is finished, then it repeats the procedure for the second layout row, and so on until it completely finishes describing the whole layout.

The output metrology deck for Hitachi metrology tool should have the same row format of the Calibre spreadsheet file. However, the arrangement of rows in the Hitachi spreadsheet file follows a different convention. It assumes the presence of SEM pattern recognition marks. Around each marker point in the layout are four test structures as shown in Figure 7-4.

Figure 7-4. SEM Recognition Marks (Hitachi Metrology Deck)



Markers in the layout are in rows and columns. The rows in the Hitachi metrology file are arranged in packets of four lines, such that the first packet is for the four test structures around the marker in the lower left corner of the layout, the second packet is for the lower second packet from the left marker, and so on until the lower row of layout markers are finished. It repeats the procedure for the next higher row packet of markers in the layout. In order to be able to generate the Hitachi metrology deck, the layout structures should have even count of rows and columns, also each marker should be surrounded by four structures; missing structures will be reported as errors.

Notes on AMAT Metrology Deck Generation

AMAT output is a .DBM file compliant with the AMAT DBM profile, version 1.5. AMAT metrology deck generation requires the AMAT CD SEM and the AMAT OPC-Check system.

The term Target in the DBM Profile corresponds to the term Test Structure as used by Calibre.

Each Target has its own XML tag.

The "Target" tag contains the following tags:

- TargetName

  The Target Name is set to be: *structurename_index_row_col*

  For example: the iso structure in row 1 col 1 in the first test pattern will be named:

  Iso_0_1_1

- TargetLocationList

  The Target Location is the X and Y coordinates on the reticle defined according to the location of the structure in the .ss file, with the transformation specified for this test pattern (rotation, mirror,...)

- TargetMetrologySpec

  The TargetMetrologySpec is defined according to the type of analysis assigned to each structure name, according to a table supplied by AMAT. If the a structure has a name that is not identified as a standard name the program will report an error.

- TargetImageList

  The TargetImage List will point to the combined gds file containing the test patterns and the coordinates will be relative to the transformation indicated.

- TargetMonitorMotivation

  The Monitor Motivation will be in the format of "OPC_MODEL_ENABLE_<1/0>_MeasWeight_<Weight>. If the structure in the ss file is set to disable, then enable flag will be 0; other wise it will be 1. The weight is the weight column in the ss file.

- TargetDeviceType

  The device type is corresponding to the location flag in the ss file "0/1" for width/space.

- TargetMaskName

  The name of the originating mask.

- TargetCoordsys\TargetCoordSysName

  The name of the originating coordinates.

- TargetCoordSys\TargetCoordSysTool

  The name of the originating CAD tool.

maskspectrum

Tools Supported: Calibre WORKbench

Description

Calculates the mask spectrum for the specified layout and bounding box. It outputs the mask spectrum data in either complex binary or real ASCII form:

- Binary file — You can load the binary file into MATLAB using the **readgrid** function.
- ASCII file — You can load the real ASCII file into WORKbench Source View.

As part of the processing, command scales *outmap* spectrum coordinates by lambda/NA.

Usage

maskspectrum -l *layout* -bbox *bboxregion* -simpixsz *size* { -outbin *binfile* | -outmap *mapfile*}

- **-l *layout***

  A layout handle, containing the geometries to be simulated. This is a required argument.

- **-bbox *bboxregion***

  A rectangular region of *layout* specified by {x1 y1 x2 y2} in database units, in which to grab the geometry used for simulations. This is a required argument.

- **-simpixsz *size***

  A value used for the size of the simulation pixel for GDS, in um. It is used to calculate number of spectrum harmonics. It must be an integer multiple of the optical diameter. This is a required argument.

- -outbin *binfile*

  A filename to store the mask spectrum data in binary format.

- -outmap *mapfile*

  A filename to store the mask spectrum data in ASCII source map (layout) format.

  Either -outbin or -outmap must be supplied as a required argument.

Example

```
set l [layout create gds/toto_modif.gds]
openlib -f setup/test.in
maskspectrum -l $l -bbox [list 0 360 900 2140] -simpixsz 0.02 \
             -outmap spectr.map
```

mergemap

Tools Supported: Calibre WORKbench

Description

Combines multiple source map files into a single averaged source map file. This is useful in averaging out errors to create a consistent source map. If you do not set the *step* option, then the step of the output map is read from the first input file. Input files can contain different steps.

You can force an output file to be symmetrical by specifying the *symtype* option:

- A value of **x** means that the output source will be made mirror-symmetrical around the X-axis.
- A value of **y** means that the output source will be made mirror-symmetrical around the Y-axis.
- A value of **xy** combines both mirrors. **xy** values is useful to make a DIPOLE illumination symmetrical.
- A value of **all** combines symmetries around axes with symmetries along both diagonals. This can be used to make symmetrical STANDARD, QUAD, and ANNULAR illuminations.

Usage

mergemap -out *fileout* -in {***file1 file2 ... filen***} [-step *step*] [-sym *symtype*]

- **-out *fileout***

  An output source map file.

- **-in** {***file1 file2 ... filen***}

  A list of input source map files. You can form the list with names in brackets "{...}" or by the Tcl "list" command "[list ...]".

- -step *step*

  An optionally specified step (pixelsize) of the output file.

- -sym *symtype*

  A keyword that specifies the type of the symmetry enforced in the output file. Can be **x**, **y, xy**, or **all**.

Example

```
% mergemap -in [list prolithex.src annular.map LUPM_75_75_50.smf] -out
out.map -step 0.01 -sym xy
```

modelfitness

Tools Supported: Calibre WORKbench

Description

Runs a simulation using the models defined in the setup file, to produce output model fitness statistics. The output statistics returned are:

Table 7-3. Model Fitness Statistics

Statistic	Description
errMean	EPE error weighted mean: $\text{errMean} = \frac{\sum W_i(EPE_{sim} - EPE_{meas})}{\sum W_i}$
errRrms	EPE error weighted root mean square: $\text{errRrms} = \sqrt{\frac{\sum W_i(EPE_{sim} - EPE_{meas})^2}{\sum W_i}}$
spec	number of EPE errors in spec (simEPE-measEPE, both LE and NLE)
errLE	max line end EPE error (simEPE - measEPE)
errNLE	max non-line end error (simEPE - measEPE)
msCorr	correlation of (modelEPE, empiricalEPE), using weights
R2	R-square coefficient of (modelThresh. empiricalThresh), using weights
adjR2	adjusted R-square of (modeThresh, empiricalThresh), using weights
rrms	threshold weighted root mean square: $\text{rrms} = \sqrt{\frac{\sum W_i(\text{thresh}_{sim} - \text{thresh}_{meas})^2}{\sum W_i}}$

The output is returned in the form of a string consisting of ordered stat/value pairs: {stat1 val1 ... statN valN}.

Usage

modelfitness -ss *ssfile* [-f *setup*] [-m *gds*] [-spec *nleSpec leSpec*]

Arguments

- **-ss *ssfile***

  A required keyword/value pair specifying the sample spreadsheet file containing the sample data.

- -f *setup*

  An optional keyword/value pair specifying the setup file to use for the simulation. This argument is only optional if a setup file has already been loaded into the application; otherwise, it is required.

- -m *gds*

  An optional keyword/value pair specifying the GDS test pattern file. This argument is only optional if a GDS file has already been loaded into the application; otherwise, it is required.

- -spec *leSpec nleSpec*

  An optional keyword and its values, used to define the allowed tolerances for the model.

  - *leSpec* defines the line-end tolerance specification in nanometers. The default is 20nm.
  - *nleSpec* defines the non-line-end (i.e. line/space) tolerance specification in nanometers. The default is 5nm.

Examples

```
% set ans [modelfitness -m tp180.gds -f setup.in -ss tp180.ss -spec 5 20]
% array set stats $ans
% puts $stats(errMean)
--> 2.6
Another run without need of specifying GDS or setup.
% set ans [modelfitness -ss tp180.ss -spec 5 20]
```

modelflow

Tools Supported: Calibre WORKbench

Description

This command creates an optimized VT5 model and an optimized optical model (or maskwriter model) using a fixed model optimization procedure that begins with a pair of "nominal" models.

Definition: *nominal* models are the initial models, specified by the setup file. In this case, the models are the optical model and the VT5 model used as a starting point for creating the optimized models.

The nominal models provide default values for parameters that are not optimized. Model Flow evaluates variations to the nominal models, as specified by the command line. It optimizes parameters in distinct optimization stages which are described in the sections that follow. The stages are performed sequentially.

- Input Arguments
- Output Arguments
- Stage 0 Arguments
- Stage 1 arguments are dependent on whether or not you are using the maskwriter functionality:
    - Stage 1 Arguments (Maskwriter)
    - Stage 1 Arguments (Optical Model)
- Stage 2 Arguments
- Stage 3 Arguments

Current modelflow Enhancements

- Support for Calibre ContourCal was added. The -cli (specifies a ContourCal information file) and -contourweight (sets the weight value of imported contour files) input arguments are optional.

Recent modelflow Enhancements

- Focus blur support has been added in stage 1, and the optical model now supports corresponding parameters. In modelflow, the following optional parameters can now be tuned (2006.3):
    - fb_lorentz_gamma and fb_lorentz_n — model chromatic errors
    - fb_gauss_sigma — model vibration errors
    - fb_rectangle_width — model tilt errors

- The Gaussian density support was expanded to handle directional (half-density) kernels by adding the options for D*k* and D*j* kernel variations to the existing D*m* and D*n* kernels in Stage 3. (2006.2)
- The **-sss** option supports calibration over a process window by enabling the use of "super spreadsheets" (multiple sample file spreadsheets concatenated together). (2006.1)
- **mtgradient** was added as a Stage 1 search type, and the **explore** option now works for mtfull, mtgradient, and mtnewton. The maximum CPU usage was raised to 32 for all mt-type searches. (2006.1)
- maskwriter optimization support was added to model mask proximity using a more-accurate triple Gaussian function; a triplet of settings (g1, g2, and g3) must be specified for this to function correctly. Note that maskwriter optimization is <u>mutually exclusive</u> with optical model operation. (2006.1)
- The **-semweight** and **-csf** options are used with the SEM Calibration tool. (2006.1)

Usage

```
modelflow { -ss spreadsheet | -sss superspreadsheet }
   -f setup {-m gds | -l layout}
     [-biasfirst ][-force][-force_symmetry]
    [-vss verification_spreadsheet]
    [-csf csffile -semweight weight]
    [-cli clifile -contourweight weight
     [output
       [log log]
       [optical file [opticalmod dir] ]
       [setup file]
       [vt5 file]
      ]
[stage0
  [modelform form]
  [nominalresist filename]
  [passfilter|excludefilter filter]
  [weight val filter f]*
]
[stage1
  [filter filter]
  [exclude_2D_geometry]
  [optics_model_1D]
  [maskwriter_model g1 begin end step g2 begin end step g3 begin end step]
  [search {full|mtfull|gradient|mtgradient|newton|mtnewton}
         [explore runcount]]
  [def_start begin end step] [num_planes begin end]
  [sigma_out begin end step] [sigma_in begin end step]
  [dsigma_out begin end step] [dsigma_in begin end step]
  [illumangle begin end step] [dillumangle begin end step]
  [imagediffusion beg end step] [kerneldiffusion beg end step]
  [beamfocus beg end step] [apodization_loss beg end step]
  [fb_lorentz_gamma {begin end step | = tcl_exp begin end}]
  [fb_lorentz_n {begin end step | = tcl_exp begin end}]
  [fb_gauss_sigma {begin end step | = tcl_exp begin end}]
```

```
    [fb_rectangle_width {begin end step | = tcl_exp begin end}]
    [apodization_loss {begin end step | = tcl_exp begin end}]
    [na beg end step]
    [imagediffusionw1 beg end step
     imagediffusions1 beg end step
     imagediffusions2 beg end step]
  ]
  [stage2 [off]
    [filter filter]
  ]
  [stage3 [off]
    [check_ref_thresh]
    [Dm [{gauss | gaussp | gaussm}] begin end step]
    [Dn [{gauss | gaussp | gaussm}] begin end step]
    [Dk [{gauss | gaussp | gaussm}] begin end step]
    [Dj [{gauss | gaussp | gaussm}] begin end step]
    [stage3a {off | copy_ref_thresh | crosscheck |nocrosscheck}]
    [stage3b {off | filter}]]
```

Input Arguments

- **-ss *spreadsheet* | -sss *superspreadsheet***

  Required argument supplying the path to the spreadsheet (-ss) or super spreadsheet (-sss) file. Calibre WORKbench uses super spreadsheets for calibration over a process window (see the section "Calibration Over a Process Window (Using Super Spreadsheets)" for more information).

- **-f *setup***

  Required argument supplying the path to the setup file.

- **-m *gds* | -l *layout***

  Required argument / value pair supplying either the path to the GDSII test pattern file or the handle of the layout containing the test pattern data.

- -biasfirst

  An optional keyword instructing the software to calculate the constant offset (epeBias) that delivers the best threshold polynomial fit.

  The -biasfirst option does not work with the **-sss** option.

- -force

  An optional argument that, when set, instructs the software to check whether or not all sites are centered in the middle of the edge. If the program encounters sites that are not centered, it exits with an error.

- -force_symmetry

  Use this optional flag to force the symmetry of the final optical model; this goal is reached by increasing the number of kernels used. (Does nothing if no optical parameters are selected for optimization in **stage1**).

- -vss *verification_spreadsheet*

  An optional argument that verifies the optimized model after Stage 3a completes, using the supplied *verification_spreadsheet.*

- -csf *csffile*

  An optional argument, specifying a calibration session file (.csf) produced by the SEMcal GUI (described in the *Calibre WORKbench User's Manual*). This argument does not need to be specified if a .csf file was previously loaded.

- -semweight *weight*

  A required argument, specifying the weight in the objective function for the SEM contours. The CD measurements from the standard spreadsheet file are given a weight of 1-*weight*. **modelflow** optimizes the total RMS *T*:

  $$T = \sqrt{wF^2 + (1-w)S^2}$$

  where -semweight is *w*, *F* is the SEM fitness function (described in the scalfit command), and *S* is the EPE RMS of the CD measurements from the sample file. When *w*=0 or if *csffile* is absent, the SEM fitness is ignored, and modelflow calibrates the model from CD measurements.

  Default is 0.25.

- -cli *clifile*

  An optional argument, specifying a contour layout information file (.cli) produced by the Calibre ContourCal GUI (described in the *Calibre ContourCal User's Manual*).

- -contourweight *weight*

  An optional argument, specifying the weight in the objective function for the SEM contours. The CD measurements from the standard spreadsheet file are given a weight of 1-*weight*, similar to using -semweight.

Output Arguments

- log *log*

  Optional argument supplying a name for the log file to create. The log file shows details of each stage of processing. Default = modelflow.log. If "stdout" is specified, the log will be written to the stdout instead of a file. The log file contains debugging information.

- optical *file* [opticalmod *dir*]

  Optional argument supplying a pathname for the output file to create for the final optical parameters text file, which Model Flow creates. The optional **opticalmod** keyword supplies a pathname for a full optical model directory. Default = none.

- setup *filename*

  Optional argument specifying a unique file name for the final setup file. The modelflow command writes the final setup file to the disk after the modelflow, and includes the final VT5 and optical models inline. Default = none.

- vt5 *file*

  Optional argument supplying a pathname for the final optimized VT5 model, which Model Flow creates. Default = none.

Stage 0 Arguments

Stage 0 performs pre-processing operations, such as setting a template model form and performing global filtering and weighting of the input data. Stage 0 is off by default, if it is not specified.

- modelform *form*

  An integer-valued index for selecting a "template" VT5 model form, which will be used instead of the nominal input VT5 model. The modelform should be an integer number. The valid forms are described in "VT5 Model Form Descriptions". Default = use form of nominal model from setup file or nominalresist statement if present.

Note

modelform must be set to 0 if you use maskwriter optimization in Stage 1.

- nominalresist *filename*

  The pathname for a VT5 model file to be substituted for the model declared in the setup file. Default = use nominal model from setup file.

- passfilter|excludefilter *filter*

  A filter for either describing which data from the spreadsheet to use, or which data from the spreadsheet to remove. Default = no filter, use all spreadsheet data.

  Filter syntax:

  <***filter***> = {*stype* | **row** *X* [**col** *Y*]}$^{+}$ **end**

  One or more structure types can be selected by the filter. The filter can match by (row, column) or by structure name or the keyword **all** to match all structures. The terminating "**end**" is required to signify the end of the filter. Some examples:

```
filter iso pitch end  // matches iso and pitch structures
filter row 8 row 9 iso end // matches iso and row 8 and row 9
```

- weight *val* filter *filter*

  The weighting values to apply to the sample points described the *filter*. You can supply as many weightings as needed. The weights are set in the order given on the command line.

You must specify filters using the syntax described for the passfilter|excludefilter filter arguments.

Default = no weight filters, use weights in spreadsheet.

Note that the weights set here are "global" and are inherited by all other processing stages. For example to set everything to weight 1, and then pitch to weight 8:

weight 1.0 filter all end

weight 8.0 filter pitch end

Stage 1 Arguments (Maskwriter)

In maskwriter mode, Calibre WORKbench performs optimization on the maskwriter model instead of the optical model. This mode uses a special optical model with customized kernels to model mask proximity, using the supplied Gaussian lengths to tune a triple Gaussian function.

Calibre WORKbench tunes the maskwriter model by changing the following parameter:

- maskwriter_model g1 *begin end step* g2 *begin end step* g3 *begin end step*

  Sets maskwriter model optimization based on a triple Gaussian convolution. The three specified Gaussian lengths (sigma) in microns are fitted to mask proximity data using a full factorial search. **modelflow** calibrates the model to CD data from the spreadsheet file (**-ss** option).

  The search interval must be specified for all three Gaussian densities.

  This optimization is off by default if it is not specified.

Stage 1 Arguments (Optical Model)

Stage 1 tunes the optical model using the empirical data. In this stage, the optical model is varied by changing various parameters. For each variation of the optical model, an optimized CTR (resist) model is used to determine the EPE rms fitness of the model. Stage 1 is off by default, if it is not specified.

- filter *filter*

  An optional spreadsheet filter for this stage only. Default = use entire spreadsheet file.

- exclude_2D_geometry

  An optional flag that generates an optical simulation, then filters out all the points that require 2D optical simulation (such as line ends and T-junctions) from the spreadsheet. The spreadsheet points that are kept for stage 1 optimization are reported in the log file.

- optics_model_1D

  An optional flag that excludes all spreadsheet points that require 2D optical simulation (you do not need to specify the exclude_2D_geometry flag) and then performs all the optimization steps of stage 1 using only 1D optical models. Since 1D optical models are very fast to generate, using this flag enables rapid optimization of the optical parameters based on the measurements from a subset of the original spreadsheet points (only the 1D data are kept, such as iso lines, lines/spaces). The spreadsheet points that are kept for stage 1 optimization are reported in the log file.

- search {full | mtfull | gradient | mtgradient | newton | mtnewton}

  Sets the optimization algorithm.

  - **full** - causes a full factorial search over all combinations of variables.
  - **mtfull** - is a multi-threaded version (uses up to 32 CPUs) of the full search algorithm.
  - **gradient** - uses a simple conjugate gradient descent method.

    Two things to keep in mind:

    - The full search and gradient search can converge to different solutions.
    - The gradient search can be orders of magnitude faster, especially for large search spaces.
  - **mtgradient** - is a multi-threaded version of the gradient algorithm (uses up to 32 CPUs and no more than 2 CPUs during the gradient search).
  - **newton** - uses a quasi-Newton optimization method, in which the second derivatives are evaluated in terms of the sequence of gradients from successive iterations using an update formula for the Hessian matrix.

    You configure your optimization process using *begin* and *end* search intervals, which must also be within the parameter boundaries as shown in Table 7-4.

Table 7-4. Search Intervals Corresponding to Modelflow Parameters for Newton Optimizations

Suggested Intervals (Begin ... End)	Modelflow Parameter Boundaries
nominal +/- 0.1	0 < sigma_out < 1
nominal +/- 0.1	0 < sigma_in < 1
nominal +/- 10	0 < illumangle < MAX=90/180
[-0.35,0.35] or +/- resist thickness for vector model	beamfocus
[-0.35,0.35] or within resist for vector model	def_start
[0,0.05]	0 < imagediffusion < 0.501
nominal +/- 0.1	0 < dsigma_out
nominal +/- 0.1	0 < dsigma_in < dsigma_out
nominal +/- 10	0 < dillumangle
nominal +/- 0.05	0 < na < 1
[0,0.1]	0 < apodization_loss < 0.501

Table 7-4. Search Intervals Corresponding to Modelflow Parameters for Newton Optimizations (cont.)

Suggested Intervals (Begin ... End)	Modelflow Parameter Boundaries
[0,0.05]	0 < kerneldiffusion < 0.501
[0,0.05]	0 < imagediffusions1 < 0.501
[0,0.25]	0 < imagediffusions2 < 0.501
[0.5,1]	0 < imagediffusionw1
nominal	1 < num_planes < 20

The *step* value is actually used as the criterion to end optimization – iterations stop if the differences between successive values of the optimization variables are less than the step value. Accordingly, the smaller the step value, the more accurate the result and the longer the execution time.

The recommended value for *step* is 0.001 for all parameters but angles; for angles it is 0.1.

EPEs obtained using a newton search are usually lower than that achieved using the gradient method.

- **mtnewton** - A multi-threaded version (uses up to 32 CPUs) of the newton mode; multithreading is invoked during exploration runs and gradient calculations.

You can constrain the number of CPUs used by the mtfull and mtnewton options by setting the environment variable:

```
setenv CWB_MT_THREAD_COUNT value
```

to a value below the maximum CPU limit (32). The multithreading options do not consume additional licenses.

Default = full.

- explore *runcount*

Used in conjunction with the search option, this is the maximum number of runs allowed in the exploration stage of the newton/mtnewton and gradient/mtgradient optimization algorithms. The exploration stage is skipped (and the **newton** and **gradient** simulations start from the nominal optical model settings) if explore is explicitly set to 0.

The exploration stage is designed based on a 3-level full factorial search: each optimization variable is set on a grid of three values corresponding to 25%, 50%, and 75% between *begin* and *end* of the search interval (step is ignored), and a full factorial search on all possible combinations is performed.

In this approach, the number of potential exploration runs (N) grows exponentially with the number of variables (M) involved, as $N = 3^M$. Specifying the explore option allows you to avoid long run times. If *runcount* < N, a partial factorial search is used. In such a case, Calibre WORKbench randomly selects from the *N* potential full factorial runs a subset of explore to be employed in the exploration stage.

The default value for explore is 128. This means that for cases involving up to 4 variables (81 experiments) the full factorial exploration design will be used, and a partial factorial design otherwise.

Update: Using **explore** with **gradient** gives different results than using gradient without explore. To use the standard **gradient** search (the method from earlier versions of Calibre WORKbench), set **explore** to 0.

- num_planes *begin end*

  Optional argument used to evaluate variations of number of planes. Default = use nominal model setting.

- def_start *begin end step*

  Optional argument used to evaluate variations of the defocus start value. The three values define the range of values to evaluate and the increment between variations. All values are specified in microns. Default = use nominal defocus start with no variations.

- na

  Optional argument specifying the numerical aperture.

- sigma_out *begin end step*

  Optional argument used to evaluate variations of sigma_out. Default = use nominal model value.

- sigma_in *begin end step*

  Optional argument used to evaluate variations of sigma_in. This argument is only relevant to ANNULAR or QUASAR illumination sources. Default = use nominal value.

- illumangle *begin end step*

  Optional argument used to evaluate variations of the illumination angle. This argument is only relevant to QUASAR illumination sources. Default = use nominal value.

- dillumangle *begin end step*
  dsigma_in *begin end step*
  dsigma_out *begin end step*

  Optional arguments used to evaluate variations of ERF profile parameters. relevant to composite sources using the ERF profile. Default = use nominal value.

- imagediffusion *begin end step*
  kerneldiffusion *begin end step*

  Optional arguments used to evaluate variations of Gaussian diffusion in either the TCCs (imagediffusion) or the kernels. Default = use nominal values.

- beamfocus *begin end step*
  apodization_loss *begin end step*
  na *begin end step*

  Optional arguments used to evaluate variations of the parameters with the same name. Default = use nominal values.

- [imagediffusionw1 *beg end step* imagediffusions1 *beg end step* imagediffusions2 *beg end step*]

  Optional keywords and arguments used to tune double Gaussian image diffusions during Stage 1 optimization. These three keywords must be supplied as a set:

  - imagediffusions1 *beg end step* **— sigma for the first Gaussian**
  - imagediffusions2 *beg end step* **— sigma for the second Gaussian**
  - imagediffusionw1 *beg end step* — weight for the first Gaussian. (Note that the weight for the second Gaussian is calculated as 1- *imagediffusionw1*)

- fb_lorentz_gamma

  Optional parameter Γ of Lorentzian focus blur.

- fb_lorentz_n

  Optional parameter n of Lorentzian focus blur.

- fb_gauss_sigma

  Optional sigma of the Gaussian blur term.

- fb_rectangle_width

  Optional width of the focal range for the tilt term of focus blur. Note that if any of the focus blur parameters have been set in the starting optical model, then these values are inherited as defaults during the modelflow optimization of the focus blur parameter(s). If the starting optical model has no focus blur, then all the terms default to zero (no blur), expect for the exponent n of the Lorentzian, which defaults to 2.8.

- apodization_loss

  Optional specifier for the apodization loss.

Stage 2 Arguments

Stage 2 optimizes the reference threshold. It is on by default.

- off

  An optional argument used to skip stage 2 optimization. In this case, the nominal reference threshold is untouched.

Note

Stage 2 must be turned off if you are performing mask writer optimization.

- filter *filter*

  An optional spreadsheet filter for this stage. The filter selects specified structures from the global spreadsheet. The spreadsheet constructed in this way is considered temporary, and is ONLY used in stage2. You must specify filters using the syntax described for the passfilter|excludefilter filter arguments. Default = use full global spreadsheet file.

Stage 3 Arguments

Stage 3 optimizes the process model. It is on by default.

- off

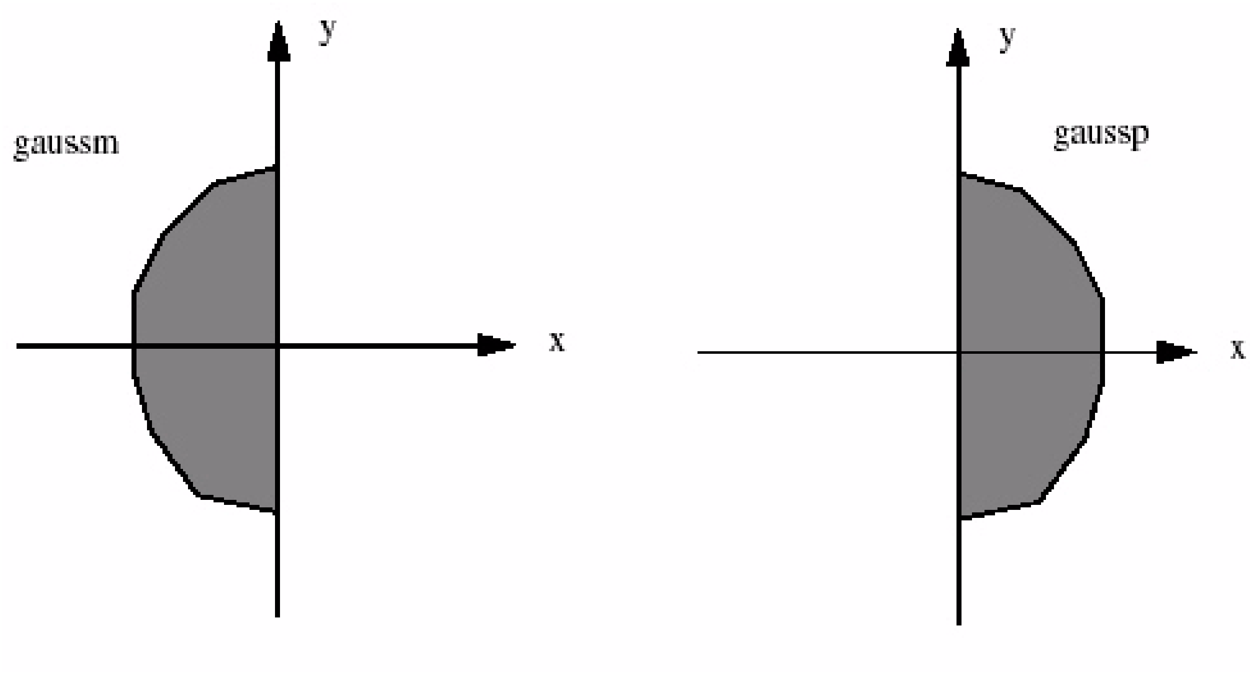
  An optional argument used to skip stage 3 optimization. In this case, the nominal threshold and bias polynomial coefficients are untouched.

- check_ref_thresh

  An optional argument that instructs the program to report an error and stop if the reference threshold does not pass through the aerial image during stage 3.   If Model Flow encounters this type of error, it reports the structure for which the violation exists.

For the following four Gaussian density kernel arguments:

The optional gauss (full density), gaussm (gauss-minus, which is 0 for positive X, and standard for X <= 0), and gaussp (gauss-plus, which is 0 for negative X, and standard for X >= 0) arguments indicate the kind of kernel that is being evaluated. The default is gauss.



Up to four regular/directional density kernels (D*m*, D*n*, D*k*, D*j*) can be optimized simultaneously. The indexes of the kernels must be unique from each other. Optimizationis performed using the full factorial search method. The valid range for *begin* and *end* values is limited by the interval [0.01, 2.01] in um.

- D*m* [{ gauss | gaussm | gaussp}] *begin end step*

  An optional argument used to evaluate variations of the VT5 gaussian density CKERNEL specified by D*m* where =1,2,..,16. If no CKERNELs are used in nominal VT5 model, then this does nothing. <u>Default = use nominal CKERNEL D1</u>.

- D*n* [{ gauss | gaussm | gaussp}] *begin end step*

  An optional argument used to evaluate variations of the VT5 gaussian density CKERNEL D*n,* where *n* != *m*, n=1,2,...,16. If D*n* is not specified in the nominal VT5 model, then this does nothing. <u>Default = use nominal CKERNEL D2</u>.

- D*k* [{ gauss | gaussm | gaussp}] *begin end step*

  An optional argument used to evaluate variations of the VT5 gaussian density CKERNEL D*k,* where *k* != *m* or *n*, *k*=1,2,...,16. If D*k* is not specified in the nominal VT5 model, then this does nothing. <u>Default = use nominal CKERNEL D3</u>.

- D*j* [{ gauss | gaussm | gaussp}] *begin end step*

  An optional argument used to evaluate variations of the VT5 gaussian density CKERNEL D*j,* where *j* != *m*, *n*, or *k*, *j*=1,2,...,16. If D*j* is not specified in the nominal VT5 model, then this does nothing. <u>Default = use nominal CKERNEL D4</u>.

- stage3a {off | copy_ref_thresh | crosscheck | nocrosscheck}

  An optional argument used to control stage 3a, in which the threshold polynomial is optimized.

  The first two settings instruct Modelflow to skip stage 3a. When skipping this stage, you have the following options:

  - **off** — use nominal threshold polynomial
  - **copy_ref_thresh** — use the reference threshold as the threshold polynomial (CTR).

  The second group of settings control how Model Flow performs the stage 3a optimization. These options are:

  - **crosscheck** — perform crosschecking in all cases
  - **nocrosscheck** — do not perform crosschecking in any case

  Crosschecking involves splitting the data in the input spreadsheet into two data sets (even and odd) that can be used to check for a stable model. If both sets will converge to the same eigenvalue the VT5 model will be stable. If they do not converge, either some of the data in one of the groups is bad or the model is not stable.

  The RMS of the VT5 model from the odd dataset is verified for the even dataset, yielding RMS1. Then the VT5 model from the even dataset is verified for the odd dataset, yielding RMS2. When the minimum eigenvalue is decreased, RMS1 and RMS2 first decrease, which indicates that this eigenvalue increases the fitness of both models on verification data, then it starts decreasing. This indicates that starting from some mineigenval the models overfit into the data and stop predicting behavior of the verification datasets.

Modelflow considers the best minimum eigenvalue to be the arithmetic average of these two values.

By default, Model Flow performs crosschecking any time the number of threshold coefficients exceeds 16.

- stage3b {off | *filter*}

  An optional argument used to control stage 3b optimization.

  - **off** — skip stage 3b and use nominal bias polynomial
  - *filter* — use a spreadsheet filter when optimizing the bias polynomial coefficients.

    Filter syntax:

    filter **=** {*stype* | **row** *X* [**col** *Y*]}$^+$ **end**

  One or more structure types can be selected by the filter. The filter can match by (row, column) or by structure name or the keyword **all** to match all structures. The terminating "**end**" is required to signify the end of the filter. Some examples:

```
filter iso pitch end  // matches iso and pitch structures
filter row 8 row 9 iso end // matches iso and row 8 and row 9
```

VT5 Model Form Descriptions

Because of the large number of model forms you can design with VT5, Mentor Graphics Corporation has developed a model naming convention and a set of basic model forms. When you use these model forms in Model Center as the basis for a new VT5, the tool fills in the data fields with appropriate default value. When you use a model form in Model Flow, it defines the basic form of the VT5 models it evaluates.

Definitions: A cross-term is a term which is the product of two image parameters, an example cross-term is (I_{max}*slope). No model forms have cross-terms between densities. Cross-terms may/may not be present between VT5 image parameters. Quadratic models with no cross terms will have linear and squared terms.

IMPORTANT: The following model options are outside the influence of specific model forms:

```
epeBias
referenceThreshold
useLogSlope
sampleSpacing
kerngrid
hoodpix
minEigenval
```

The experimental models inherit all these options from the nominal VT5 model. You should ensure that the nominal VT5 contains the desired settings for these options. You can examine the Model Flow output log to check which options were inherited by the final VT5 model.

Model Form Naming Conventions

The modelform number forms an encoded description of the model characteristics. Each model form identifier has three digits:

XYZ

Example: For model form 23:

```
X = 0
Y = 2
Z = 3
```

- X digit — Indicates the type of model.
  1 for BIAS model
  0 (or omitted) for THRESHOLD model
- Y digit — Indicates the image parameters used.

Table 7-5. Interpreting the Y Digit

value	description
0	CTR
1	Imax, slope, factor (VTRE)
2	Imax, slope (VTR)
4	slope (fast runtime)
6	slope, IMAX, IMIN
8	slope, IMAX, IMIN, factor

- Z digit — Describes the polynomial in terms of type of kernels, quadratic versus linear, and crossterms or no crossterms.

Table 7-6. Interpreting the Z Digit

value	description
0	quadratic, no densities
1	quadratic, no cross terms, no densities
2	quadratic, 2 gaussian densities
3	quadratic, no cross terms, 2 gaussian densities
4	quadratic, 10 tophat kernels

Table 7-6. Interpreting the Z Digit

value	description
5	quadratic, no cross terms, 10 tophat kernels
6	linear, 10 tophat kernels

Existing Model Forms

Table 7-7 lists the pre-defined model forms. When you use these model forms in Model Center as the basis for a new VT5, the tool fills in the data fields with appropriate default value.

Table 7-7. Pre-Defined Model Forms

	Number	Description
	0	CTR Model
VTRE (Imax, slope, factor)	10	quadratic (similar to VTRE 322)[1]
	11	quadratic, no cross terms
	12	quadratic, 2 densities
	13	quadratic, no cross terms, 2 densities
VTR (Imax, slope)	22	quadratic, 2 densities
	23	quadratic, no cross terms, 2 densities
	24	quadratic, 10 tophat kernels
	25	quadratic, no cross-terms, 10 tophat kernels
	26	linear, 10 tophat kernels
slope only (for fast runtimes)	43	quadratic, no cross terms, 2 densities
	45	quadratic, no cross-terms, 10 tophat kernels
	46	linear, 10 tophat kernels
slope, IMAX, IMIN	61	quadratic, no cross terms
	63	quadratic, no cross terms, 2 densities
	65	quadratic, no cross-terms, 10 tophat kernels
	66	linear, 10 tophat kernels
slope, IMAX, IMIN, factor	81	quadratic, no cross terms
	83	quadratic, no cross terms, 2 densities
	85	quadratic, no cross-terms, 10 tophat kernels
	86	linear, 10 tophat kernels

Table 7-7. Pre-Defined Model Forms

Bias Model Forms	as above, +100	Model form numbers between 110 and 199 represent BIAS models. The only valid model form numbers for BIAS models are those that correspond to a valid THRESHOLD model. For example, 145 is a valid model form: • Model form 45 = slope only, quadratic, no cross-terms, 10 tophat kernel threshold polynomial • Model form 145 = slope only, quadratic, no cross-terms, 10 tophat kernel bias polynomial There are no BIAS model forms analogous to model form 0.

1. A VT5 model will never be exactly equivalent to a VTRE model because some parameters are calculated differently (that is, based on a reference threshold and reference image.)

Note

The model form you use for your process may impact the way you develop your setup file. For example, you would not use visible layers for OPC with Printed Assist Features if your model form is one of the VT5 models that have density kernels (12, 13, 43, 63, 83, 112, 113, 143, 163, and 183).

OPC with Printing Assist Features (PAFs) performs OPC on assist features you want printed (such as dummy fill for density assistance). You must code your setup file to handle the PAFs.

A longer discussion of how to perform these advanced OPC methods is discussed in the *Calibre Model-Based OPC User's Manual*, in the section "OPC Scenarios."

Dependent and Independent Optical Model Variables

There can be two types of optimization variables associated with the optical model: independent and dependent.

- Variations of an **independent** variable are determined by three (two in the case of num_planes) numbers (*begin*, *end*, and *step)* which specify the search interval and the step size.
- For example:

```
sigma_out 0.6 0.8 0.05
```

- Variations of a **dependent** variable are determined by a Tcl expression and two numbers (*Tcl_exp*, *begin*, and *end*); the two numbers specify the search interval, while the Tcl expression specifies the relationship between this variable and the other independent variables involved in optimization.

- For example:

  ```
  sigma_in = {$sigma_out - 0.2} 0.4 0.6
  ```

  The Tcl expression must be enclosed in curly brackets and can only include independent variables (no dependent variables and/or parameters that are not involved in optimization are allowed in this expression). If the value of a dependent variable obtained using the associated Tcl expression appears to be outside of the search interval, this variable will be set to the nearest boundary value (either *begin* or *end*).

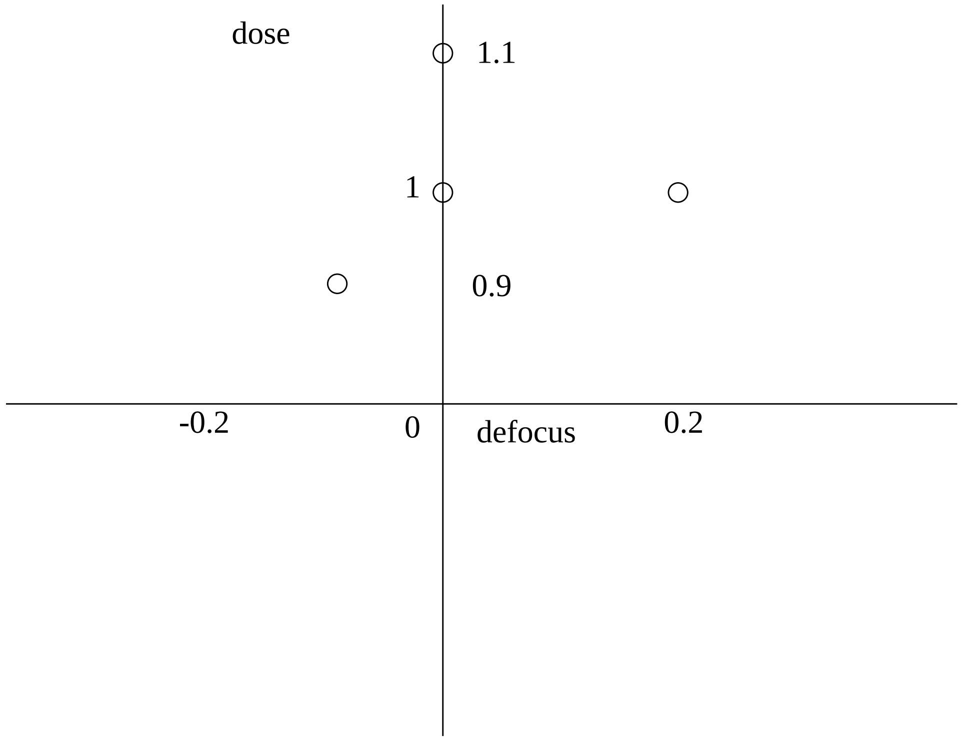
Calibration Over a Process Window (Using Super Spreadsheets)

The main purpose of calibration over a process window (also referred to as "PW Calibration") is to build more robust optical and (especially) VT5 models. Sampling over a range of process conditions widens the IMAX, SLOPE, and other ranges and makes VT5 models more stable.

The general recommendation is to collect data from 3 to 4 process conditions, such as 2 different defocus and 2 different dose conditions.

PW Calibration is conducted over spreadsheet CD data collected under varying process conditions of the defocus and dose. Defocus and dose can be sampled in an arbitrary manner (shown as the circles in Figure 7-5), not necessarily using fixed deltas, and/or covering all corners of the process window.

Figure 7-5. Sampling Example



Define the following terms:

- “defocus” is an offset from the best focus conditions in *um*. In simulations, it is added to the “beamfocus” (when “beamfocus” is present in the nominal model), or to the “defocus_start” parameter of the optical model otherwise.
- “dose” is a dimensionless parameter specifying the dose ratio to the best dose. It is mapped during simulations to the setup file “dose” for the first mask (in other words, the setup file dose is multiplied by this “dose”).

The user has to collect all relevant spreadsheet (.ss) files, one per process window sampling. Each .ss file can sample CDs from different locations (the number of lines in each .ss file is independent of other .ss files). However, all locations have to be from the same GDS file.

You concatenate the .ss files into one super spreadsheet file that is passed to the modelflow command, specifying the super spreadsheet file to the -sss option:

```
modelflow … -sss superspreadsheet …
```

The use of super spreadsheets has the following additional limitations:

- Using the Stage 3 check_ref_threshold option picks the spreadsheet at the best process conditions (dose = 1, defocus = 0). If the super spreadsheet does not contain a spreadsheet at those process conditions and you specify check_ref_threshold, an error message is generated.
- Similarly, if you use the -csf option, you must have a spreadsheet containing best process conditions, since SEM contours are assumed to be taken at dose = 1, defocus =0.
- Different spreadsheet files inside the super spreadsheet can refer to the same process conditions, but this is not recommended, since it may increase the runtime slightly. The spreadsheet with the best process conditions are not merged; only the first one containing dose = 1, defocus = 0 is used.

Examples

Gradient Search Involving SEM Calibration File Example

In this example, we tune several optical model parameters to fit a constant threshold resist model using data from a spreadsheet file and a SEM Calibration file. The number of exploration runs is set to zero and therefore the gradient search starts with nominal optical settings. After 42 iterations, the optimization process has converged to a model whose fitness is better by a factor of 2 than that of the nominal model

```
modelflow -ss weighted.ss -csf sem.csf -semweight 0.4 \
  -m tp.gds -f setup.in \
  output log out.log \
  stage0 modelform 0 \
  stage1 \
    filter pitch end \
    search gradient explore 0 \
    sigma_out .7 .9 .02  \
    sigma_in .3 .54 .02  \
    illumangle  25 35 1 \
    dsigma_out 0 .1 .01  \
    dsigma_in 0 .1 .01 \
    dillumangle 0 10 1 \
    imagediffusion 0 .1 0.01
```

Following are excerpts from the log file:

```
-->NUMBER OF DIMENSIONS FOR GRADIENT SEARCH = 7
-->OVERALL SIZE OF SPACE = 23030293

BEGIN STAGE 1 OPTIMIZATION RECORD 0 (nominal)
--> EPE RMS = 4.009
SIGMA_OUT       0.8000
SIGMA_IN        0.5000
ILLUMANGLE     30.000
D_SIGMA_OUT     0.0500
DILLUMANGLE     5.0000
D_SIGMA_IN      0.0500
imagediffusion  0.0000
END STAGE 1 OPTIMIZATION RECORD
...
BEGIN STAGE 1 OPTIMIZATION RECORD 42
--> EPE RMS = 1.765
SIGMA_OUT            0.9000
SIGMA_IN             0.3600
ILLUMANGLE          35.0000
DILLUMANGLE          8.0000
D_SIGMA_OUT          0.0300
D_SIGMA_IN           0.0900
imagediffusion       0.0300
END STAGE 1 OPTIMIZATION RECORD

-->BEST STAGE 1 EPE RMS = 1.765

SIGMA_OUT          0.9000
SIGMA_IN           0.3600
ILLUMANGLE        35.0000
DILLUMANGLE        8.0000
D_SIGMA_OUT        0.0300
D_SIGMA_IN         0.0900
imagediffusion     0.0300
```

If we compare this result to the best RMS obtained by varying degrees of tuning, we see the sensitivity of final EPE RMS to various tuning parameters.

Table 7-8. Comparison of Tuning Runs

params	**error (RMS)**
sigma, sigma_in, dsigma, dsigma_in, illumangle, dillumangle, imagediffusion	1.77 nm
sigma, sigma_in, dsigma, dsigma_in, illumangle, dillumangle	1.95 nm
sigma, sigma_in, imagediffusion	2.04 nm
sigma, sigma_in	2.20 nm
no tuning	4.01 nm

Maskwriter Model Optimization Example

In this example, we optimize diffusion lengths of three Gaussian density kernels to simulate a mask layout bypassing the optical model.

```
modelflow \
  -f setup.in \
  -m tp.gds \
  -ss sample_final.ss \
  output \
    log out.log \
  stage0 \
    modelform 0 \
    weight 1 filter all end \
    weight 4 filter pitch end \
    weight 2 filter iso end \
  stage1 \
    maskwriter_model g1 0.01 0.05 0.01 g2 0.04 0.2 0.04 g3 0.2 0.5 0.1 \
  stage2 off
```

VT5 With Gaussian Half-densities Example

In this example, we optimize diffusion lengths of two regular and two directional (gaussm and gaussp) density kernels. Note that for the half-densities to be properly rotated, you must also set the environment variable LITHO_VT5_DIRECTIONAL_DENSITY to be 1.

```
modelflow \
  -ss test.ss  \
  -m tp.gds  \
  -f setup.in  \
  output \
    log out.log \
  stage2 \
    off  \
  stage3 \
    D1 0.05 0.15 0.1 \
    D2 gauss  0.38 0.48 0.1 \
    D6 gaussm 0.16 0.26 0.1 \
    D3 gaussp 0.27 0.37 0.1 \
  stage3a off
```

odanalyze

Tools supported: Calibre WORKbench

Description

Analyzes errors induced in the optical model from truncating the optical diameter.

The command requires a test layout containing only 1D test geometries and an associated spreadsheet file. It calculates the CDs based on a CTR model for a very large optical diameter (baseline - default = 4um), followed by re-calculating each optical diameter in the specified search range (using the -range switch). Optionally, it can also calculate and report the intensity values along each simulation site and each optical diameter setting.

Note that if any structure in the test layout is detected as not one-dimensional, this command returns an error.

Usage

odanalyze -ss *ssfile* **-range** *od_search_range* [-l *layout* | -m *gds*] [-f *setup*] [-o *output*]
[-step *od_step*] [-odbase *od_base*] [-outputI 0|1]

Arguments

- **-ss *ssfile***

  A required spreadsheet file with coordinates where simulations will be performed.

- **-range *od_search_range***

  A required Tcl string that defines the starting and ending value of the optical diameter range.

- -l *layout*

  Optional layout handle with test pattern. Default: The loaded layout is used.

- -m *gds*

  Optional gds file with test pattern. Default: The loaded layout is used.

- -f *setup*

  Optional setup file. Default: The loaded setup file is used.

- -o *output*

  Optional output file with simulation results. Default: The results are written into text file "odanalyze.log"

- -step *od_step*

  Optional step size in um to cover the search range *od_search_range*. Default is 0.1um.

- -odbase *od_base*

  Optional optical diameter (in um) for baseline simulations. Default is 4um.

- -outputI 0|1

  Optional flag to output intensity values along the simulation sites. Default is 0.

Return Value

The return string contains records in the following format:

```
OD optical_diameter_in_um SAMPLE sample_num CD_WAFER simulated_CD
DCD_WAFER delta_CD_compared_to_baseline_CD CD_DRAWN CD_from_spreadsheet
TYPE 0_or_1_from_spreadsheet COORDS x1_y1_x2_y2 EPE1
1st_edge_simulated_EPE EPE2 2nd_edge_simulated_EPE
```

The coordinates (COORDS record) are the coordinates of the edge centers. Multiple records are separated by “:”. Note that multiple records arise because you have simulations for multiple optical diameters and multiple sample points, so the records span a two-dimensional matrix of simulation results.

Example

In the following example we run odanalyze on a layout containing two test patterns: one isolated line and one 1:1 line/space pattern. The baseline optical diameter is 8um, and the search range is from 1.0um to 1.6um in steps of 0.2um:

```
%odanalyze -f setup/setup1.in -m gds/example1.gds -ss sample/example1.ss
-range "1.0 1.6" -step 0.2 -odbase 8.0 -outputI 1
```

The following is the beginning of the return string, containing the first two records that appear on the screen (the whole string can be very large):

```
OD 8.000000 SAMPLE 1 CD_WAFER 103.304343 DCD_WAFER 0.000000 CD_DRAWN
120.000000 TYPE 0 COORDS 59940 60000 60060 60000 EPE1 -8.347774 EPE2 -
8.347883:OD 8.000000 SAMPLE 2 CD_WAFER 99.466623 DCD_WAFER 0.000000
CD_DRAWN 120.000000 TYPE 0 COORDS 99940 60000 100060 60000 EPE1 -10.265685
EPE2 -10.267692:   ...
```

The output results are also logged in the (default) file *odanalyze.log*. Note that the intensities are also included in the output, since the flag *-outputI* was set.

```
%more odanalyze.log
------------------------------------------------------------------------
-------
-----
-----  BASELINE OPTICAL DIAMETER - OD = 8 um
-----
------------------------------------------------------------------------
-------
SAMPLE 1 CD_WAFER 103.304343 DCD_WAFER 0.000000 CD_DRAWN 120.000000 TYPE 0
COORDS 59940 60000 60060 60000 EPE1 -8.347774 EPE2 -8.347883 INTENSITY
0.213165 0.248565 0.286770 0.326806 0.369794 0.412699 0.458641 0.502586
0.549569 0.592786
SAMPLE 2 CD_WAFER 99.466623 DCD_WAFER 0.000000 CD_DRAWN 120.000000 TYPE 0
COORDS 99940 60000 100060 60000 EPE1 -10.265685 EPE2 -10.267692 INTENSITY
0.230079 0.264446 0.301995 0.339035 0.379259 0.416346 0.456618 0.491107
0.528777 0.558196
------------------------------------------------------------------------
-------
-----
-----  OD = 1 um
-----
```

```
------------------------------------------------------------------------
-------
SAMPLE 1 CD_WAFER 102.851805 DCD_WAFER -0.452538 CD_DRAWN 120.000000 TYPE
0 COORDS 59940 60000 60060 60000 EPE1 -8.580524 EPE2 -8.567671 INTENSITY
0.214699 0.250242 0.288594 0.328771 0.371907 0.414902 0.460939 0.504847
0.551800 0.595000
SAMPLE 2 CD_WAFER 99.214425 DCD_WAFER -0.252198 CD_DRAWN 120.000000 TYPE 0
COORDS 99940 60000 100060 60000 EPE1 -10.389866 EPE2 -10.395709 INTENSITY
0.232121 0.265867 0.302887 0.339524 0.379459 0.416317 0.456470 0.490770
0.528360 0.557733
------------------------------------------------------------------------
-------
-----
-----  OD = 1.2 um
-----
------------------------------------------------------------------------
-------
SAMPLE 1 CD_WAFER 102.851805 DCD_WAFER -0.452538 CD_DRAWN 120.000000 TYPE
0 COORDS 59940 60000 60060 60000 EPE1 -8.580524 EPE2 -8.567671 INTENSITY
0.214699 0.250242 0.288594 0.328771 0.371907 0.414902 0.460939 0.504847
0.551800 0.595000
SAMPLE 2 CD_WAFER 99.214425 DCD_WAFER -0.252198 CD_DRAWN 120.000000 TYPE 0
COORDS 99940 60000 100060 60000 EPE1 -10.389866 EPE2 -10.395709 INTENSITY
0.232121 0.265867 0.302887 0.339524 0.379459 0.416317 0.456470 0.490770
0.528360 0.557733
------------------------------------------------------------------------
-------
-----
-----  OD = 1.4 um
-----
------------------------------------------------------------------------
-------
SAMPLE 1 CD_WAFER 103.287141 DCD_WAFER -0.017202 CD_DRAWN 120.000000 TYPE
0 COORDS 59940 60000 60060 60000 EPE1 -8.344928 EPE2 -8.367931 INTENSITY
0.213080 0.248501 0.286733 0.326813 0.369851 0.412797 0.458781 0.502741
0.549735 0.592918
SAMPLE 2 CD_WAFER 99.283568 DCD_WAFER -0.183055 CD_DRAWN 120.000000 TYPE 0
COORDS 99940 60000 100060 60000 EPE1 -10.349234 EPE2 -10.367198 INTENSITY
0.230585 0.265007 0.302628 0.339713 0.379998 0.417098 0.457394 0.491829
0.529459 0.558756
------------------------------------------------------------------------
-------
-----
-----  OD = 1.6 um
-----
------------------------------------------------------------------------
-------
SAMPLE 1 CD_WAFER 103.287141 DCD_WAFER -0.017202 CD_DRAWN 120.000000 TYPE
0 COORDS 59940 60000 60060 60000 EPE1 -8.344928 EPE2 -8.367931 INTENSITY
0.213080 0.248501 0.286733 0.326813 0.369851 0.412797 0.458781 0.502741
0.549735 0.592918
SAMPLE 2 CD_WAFER 99.283568 DCD_WAFER -0.183055 CD_DRAWN 120.000000 TYPE 0
COORDS 99940 60000 100060 60000 EPE1 -10.349234 EPE2 -10.367198 INTENSITY
0.230585 0.265007 0.302628 0.339713 0.379998 0.417098 0.457394 0.491829
0.529459 0.558756
```

openlib

Tools Supported: Calibre WORKbench, LITHOview

Description

Loads a GDS file, a setup file, or both into the Calibre WORKbench or LITHOview application. Once the setup file is loaded, it remains active until another setup file is loaded by **openlib** or any of the other commands that allow you to load a setup file, such as **samlinev2**.

Usage

openlib -m *gds* -f *setup*

Arguments

- **-m *gds***

  A keyword/value pair specifying the path for the GDS file to load into the Calibre WORKbench application. You must specify either **-m** or **-f** or both.

- **-f *setup***

  A keyword/value pair specifying the path for the setup file to load into the Calibre WORKbench application. You must specify either **-m** or **-f** or both.

Note

To find out which setup file is currently loaded, issue the command **gcsuf** at the Tcl prompt.

opticsgen

Tools Supported: Calibre WORKbench

Description

Creates an optical model and associated source map file. Prior to invoking this command, you must create an Optical Parameters file containing the parameters that characterize the model. For information on the format of the optical parameter file, refer to "Optical Parameters File Format" on page 342.

Usage

opticsgen -opticalfile *optical* -o *outmodel*

Arguments

- **-opticalfile *optical***

  A required keyword/value pair specifying the optical parameter file that defines the optical model.

- **-o *outmodel***

  A required keyword/value pair specifying the name of the resulting optical model. The application creates a directory with this name and stores all the files required for the model in this directory, including:

  a. binary files with the lookup-tables for the SOCS kernel / mask convolutions

  b. illummap.smf - source map text file, which you can view using the Source View tool in Calibre WORKbench

Note

When specifying any filename, you must be sure to provide either the absolute path *or* the path relative to the directory from which you invoke the Calibre WORKbench application.

You can instruct this command to write the optical kernels to disk by setting the environment variable WB_PRINT_MODEL. Use:

```
setenv  WB_PRINT_MODEL  1       in the .cshrc file or
set env(WB_PRINT_MODEL) 1       in the .signaext file or tclsh command prompt
```

This causes the kernels to be written to files h0.bin, h1.bin, etc.

You can read these files using the following Matlab script:

```
% y = readgrid(fname);
% Returns an array from the gridhead in the file fname
function [t] = readgrid(fname)
fid = fopen(fname,'r');
info = fread(fid, 3 ,'int');
sz = [info(1) info(2) ]
```

```
type = info(3)
% reading the pointer from the gridhead
useless = fread(fid, 4, ’char’);
if type == 1
  t = fread(fid, sz, ’float’);
  t = t’;
else
  t2 = fread(fid,[ 2*sz(1) sz(2)],’float’);
  clear i;
  t2 = t2’;
  for n2 = 1:sz(2),
    for n1 = 1:sz(1),
      t(n2,n1) = t2(n2,2*n1 - 1) + i*t2(n2,2*n1);
    end
  end
end
fclose(fid);
```

resistgenv2

Tools Supported: Calibre WORKbench

Description

Creates a resist model to fit the data contained in the sample spreadsheet and the samdata file.

resistgenv2 uses the currently loaded model as a template for the model it creates. From this template model, it extracts the model type (VTRE, VTR, or CTR) and the model parameters (M, N, C, log slope options, sample Spacing, and so on). This template model must be the same resist model that was loaded (via the setup file) for generating the samdata file using samlinev2.

Requirements

Before issuing this command you must:

- Load a setup file using openlib.
- Create a samdata file using samlinev2.

Usage

resistgenv2 *samdata ssfile out.mod*

Arguments

- ***samdata***

  A required argument specifying the sample data file that contains the simulation data for the test pattern.

- ***ssfile***

  A required argument specifying the sample spreadsheet file containing the empirical data for the test pattern.

- ***out.mod***

  A required argument specifying the name of the model to create.

Note

When specifying any filename, you must be sure to provide either the absolute path *or* the path relative to the directory from which you invoke the Calibre WORKbench application.

samlinev2

Tools Supported: Calibre WORKbench

Description

Simulates the printed image at all sites represented in the sample spreadsheet. This command generates an output file, called the Sample Data File, that characterizes the aerial image in terms of Imax, slope, and so on.

When interpreting the output from samlinev2, remember that the definitions of some parameters vary according to the type of resist model specified in the setup file. For example, Imax and Imin using VT5 models are calculated within a slightly different range than using VTRE models. For a complete discussion of the parameter definitions, refer to Table 8-2.

Note

For this command to work properly, the sitelength must be larger than 0.64 um, sitespacing should be less than 0.06 um, and site type must be SAMPLE.

Usage

samlinev2 -ss *spreadsheet* -f *setup* {**-m *tp_file* | -l *handle*** } [-o *samdata*] [-sso *cleanss*]
[-check_ref_thresh] [-force] [-check_site *enableflag*]

Arguments

- **-ss *spreadsheet***

  A required keyword/value pair specifying the sample spreadsheet file containing the empirical data for the test pattern.

- **-f *setup***

  A required keyword/value pair specifying the setup file to load into the Calibre WORKbench application.

- **-m *tp_file* | -l *handle***

  The test pattern file or a layout object handle (cannot use both).

- -o *samdata*

  An optional keyword/value pair specifying the name of the output file. This file contains the simulated aerial image data for the test pattern, which includes Imax, slopes, etc. The default is *samdata.txt*.

- -sso *cleanss*

  An optional argument that instructs samlinev2 to output a “cleaned” spreadsheet file which is similar to the input spreadsheet file, but all lines that do not intersect a reference threshold are removed. Calibre WORKbench writes the file to the specified filename.

- -check_ref_thresh

  An optional flag. When specified, instructs the program to print an error message to the log file if the resist model requires a reference threshold (that is, it is a VT5model) and it encounters a sample structure whose intensity does not pass through the reference threshold.

- -force

  An optional flag. When specified, "forces" the program to compute at all sites, regardless of whether they are centered on a fragment or not. Normally, any site that is not centered on a fragment is considered an error to be flagged.

- -check_site *enableflag*

  An option that controls whether or not the command checks that the sites are type SAMPLE and are long enough to overlap with the empirical EPE before running the command. Allowed values for *enableflag* are:

  - 1 — Checks sites. If sites are not of type SAMPLE, or if any site is not long enough, the samlinev2 operation generates an error, which it writes to the transcript.
  - 0 — Does not check sites. Use this setting if you know the sites are properly defined.

  The default value is 1.

Note

When specifying any filename, you must be sure to provide either the absolute path *or* the path relative to the directory from which you invoke the Calibre WORKbench application.

scalfit

Description

Calculates the fitness value of the specified SEM image. You use this command in the SEM calibration phase of development.

It returns a string in the format:

`-fit` *value1* `-count` *value2* `-threshold` *value3*

when the mode is aerial, and a string in the format:

`-fit` *value1* `-count` *value2*

when the mode is dense.

- The -fit value is the fitness metric in nm.
- The -count value is number of sites that have been used in fitness calculation.
- The -threshold value is aerial image best threshold value to fit SEM contours.

The fitness objective function is the average fitness of all the SEM pictures. Each picture is measured for its contour difference.

The information calculated by the fitness function can be stored in a layout *handle* if one is set in the **-deblayout** option. The "green," simulated contours, EPEs as text, and vectors are dumped into the specified layout.

Note that the iteration over the green line and finding distances to the dense printimage contour uses a robust algorithm that automatically skips potential gaps in the SEM contours, and is insensitive to the lengths of the dense printimage edges.

Usage

scalfit [-csf *csffile*] [-f *setup*] [-mode dense | aerial] [-deblayout *handle*]
[-deblayer *layernum*] [-debcell *cellname*] [-debtype <u>path</u> | poly]

Arguments

- -csf *csffile*

  An optional argument, specifying a calibration session file (.csf) produced by the SEM Calibration GUI. This argument does not need to be specified if a .csf file was previously loaded.

- -f *setup*

  An optional argument, specifying a Calibre WORKbench setup file. This argument does not need to be specified if the setup file is already loaded.

- -mode dense | aerial

  An optional argument, defining the simulation mode. It runs dense printimage simulations in dense mode. In aerial mode, this command runs an aerial image simulation to get the initial aerial image grid, then examines different slices of the aerial image grid to find the optimal threshold. This mode is used when finding reference threshold.

- -deblayout *handle*

  An optional argument, specifying an output layout handle for debugging of the fitness function.

- -deblayer *layernum*

  An optional argument, specifying the layer number to start exporting the debugging information at. Each SEM picture uses a set of four layers:

  - Layer N is used for the “green” contour
  - N+1 is used for simulated contours
  - N+2 is used to display edges that represent vectors from the green contour to the simulated contour
  - N+3 is used to display fitness values as text (EPE in nm), which are the per point contributions to the fitness metric. The default value is 83.

- -debcell *cellname*

  An optional argument specifying the layout cell name for output. The default is “TOPCELL.”

- -debtype path | poly

  An optional output specifying whether to output “green” contours as polygons or paths. The default is path.

Example 1

```
#Compare dense and aerial mode fitnesses
array set res [scalfit -mode aerial -csf poly.csf -f poly.in]
array set res2 [scalfit -mode aerial -f poly2.in]
array set res3 [scalfit -mode dense]
puts “Aerial simulations fit: $res(-fit)”
puts “Aerial simulations best threshold: $res(-threshold)”
puts “Fit for second setup file, aerial: $res2(-fit)”
puts “Fit for second setup with VT5: $res3(-fit)”
```

Example 2

```
#Writes debugging information into layers 20+
set l [layout create]
$l create cell TOP
array set ares [scalfit -debtype poly -mode dense -csf poly.csf -deblayout
$l -deblayer 20 -debtype path -f poly5_1.in]
$l oasisout green.oas
```

scalcontour

Description

This command calculates and exports the SEM contour (“green” contour) to the layout object. The SEM contours depend on the SEM threshold value and curve fitting algorithms.

Usage

scalcontour -outlayout *handle* -outlayer *layernum* [-csf *csffile*] [-f *setup*]
[-outcell *cellname*] [-picno *picturenum*] [-algorithm *algorithm*] [-thresh *thresh*]
[-outtype path | poly]

Arguments

- **-outlayout** *handle*

  A required argument specifying the layout handle to output a SEM contour (“green” contour) to.

- **-outlayer** *layernum*

  A required argument specifying the layer number to output SEM contour to.

- -csf *csffile*

  An optional argument, specifying a calibration session file generated in the SEM Calibration GUI. The file does not need to be specified if it was previously loaded in scalfit or a previous call to scalcontour.

- -f *setup*

  An optional argument, specifying a Calibre WORKbench setup file. This argument does not need to be specified if the setup file is already loaded.

- -outcell *cellname*

  An optional argument specifying the cell name to output SEM contour to. The default is “TOPCELL.”

- -picno *picturenum*

  An optional argument, specifying the SEM picture number which will be used to extract the SEM contour. Starts from 1. Default is 1.

- -algorithm *algorithm*

  An optional argument, specifying the type of curve fitting algorithm that is used for the SEM contour. Can be Linear, Cubic, Cross, Thresh, Fermi-D, or Slope. The default value is read from the previously or currently loaded session .csf file.

- -thresh *thresh*

  An optional argument, specifying the SEM threshold value to extract the contour from. Must be between 0 and 1. The default value is read from the previously or currently loaded session .csf file.

- -outtype path | poly

  An optional output specifying whether to output “green” contours as polygons or paths. Default is path.

Example

```
set l [layout create]
$l create cell TOP
scalcontour -csf poly_one.csf -f poly.in -outlayout $l -outlayer 1
#Changing here SEM threshold and algorithm from
#the values which are set in poly_one.csf
scalcontour -outlayout $l -outlayer 2 -thresh 0.4 -algorithm Fermi-D -
outtype poly
$l oasisout green.oas
#Comparing to the baseline results
catch {exec diff green.oas green.oas.ref} diff_result
if { $diff_result == "" } {
puts PASS
} else {
puts FAIL
}
```

ss_markup_layout

Tools Supported: Calibre WORKbench

Description

This command goes through the sample spreadsheet and creates boxes in the layout at the locations indicated in the spreadsheet. These boxes are created on a new layer that is chosen by the application, in the top cell. The boxes are 10x10 database units wide and are useful for checking where the sites are being placed**.** Figure 7-6 shows a sample point “marked” in the layout by inserting a rectangle at the appropriate location.

Requirements

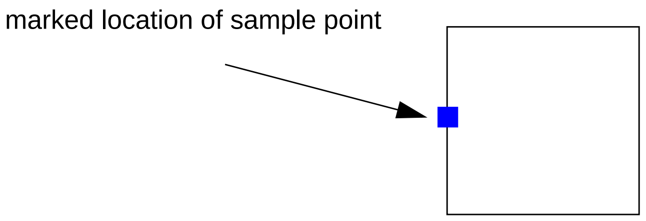
Requires Tk, so you must invoke the application in one of the GUI modes.
Before issuing this command you must:

- Load a layout file: layout create

Example

```
ss_markup_layout layout0 sample.ss
```

Figure 7-6. Box Drawn Around Off-centered Site



Usage

ss_markup_layout ***handle ssfile***

Arguments

- ***handle***

  The handle for the layout to be “marked” using the spreadsheet data.

- ***ssfile***

  A spreadsheet with sample points to write into the layout.

ss2gauge

Tools supported: Calibre WORKbench

Description

Produces a Tcl gauge object from the specified spreadsheet. The spreadsheet file must be in the following format:

```
structure_name row column x y loc drawn other meas weight
```

Caution

In 2006.4, **ss2gauge** was modified to read sample spreadsheet files in as if it were for visible etch format (used by the dense modeling tools) by specifying the -etch argument. The sample spreadsheet format has not been changed in order to maintain backwards compatibility.

Tip: For more information on gauge objects and the relationship between the various gauge commands, see the section "Gauge Object Command Relationships" in Chapter 8, "WORKbench File Formats."

Usage

ss2gauge -ss *spreadsheet* [-l *wblayout* | -m *layoutfile*] **-layer *layerno*** [-extra *extra*]

[-force] [-mode *mode count dist*][-etch][-search *distance*]

Arguments

- **-ss spreadsheet**

  A required argument specifying a spreadsheet filename to convert.

The -l and -m arguments are mutually exclusive, and one of the two must be specified.

- -l *wblayout*

  An optional argument specifying a previously-loaded Calibre WORKbench layout name.

- -m *layoutfile*

  An optional argument specifying a layout file to load, in one of the supported formats (such as GDS or OASIS).

- **-layer *layerno***

  A required argument specifying the layer number where the measurements are to be done.

- -extra *extra*

  A required argument specifying that the gauge must be lengthened by 2****extra*** (in database units). Default value is 200.

- -force

  An optional argument that forces Calibre WORKbench to produce possibly bad gauges (such as gauges not centered on the nearest edge).

- -mode *mode count dist*

  An optional argument, specifying the mode of operation, dependent on the following values:

  - *mode*- one of 0, 1, 2 or 3, corresponding to normal, average, min or max measurements. The mode specifies how CD/space measurements are done if subgauges are present.
  - *count* - number of subgauges added to the left and right of the main gauge (the total number of subgauges will be 2**count* +1).
  - *dist* - distance between subgauges, in database units.

  The default value is to create one gauge and use normal mode for measurement.

- -etch

  An optional argument. Switches the ss2gauge command to treat the sample spreadsheet file as a visible etch bias (VEB) file. This transfers the data into the appropriate etch fields in the gauge object instead of into the measured resist fields (writing to the measured resist fields is the default).

  Information on visible etch bias files can be found in the *Dense Modeling Reference*.

- -search *distance*

  An optional argument, specifying the search distance radius in microns; this is the area around a spreadsheet measurement that will be searched for the edges crossed by the gauge. (default: 0.5 um)

Example

This example is for a flat database, and creates the gauge object “aobj” from the specified GDS file and spreadsheet. It creates the gauges from the objects on layer 0, and lengthens the gauge by 20 dbu (2*10).

```
set aobj [ss2gauge -ss sram.ss -m “sram.gds” -layer 0 -extra 10]
```

This example is for a hierarchical database, which requires the layout to be flattened before gauge objects can be created:

```
set l_obj [layout create sram_hier.gds]
layout copy $l_obj l_obj_flat
set aobj [ss2gauge -ss sram.ss -l l_obj_flat -layer 0 -extra 10]
```

ssclean

Tools Supported: Calibre WORKbench

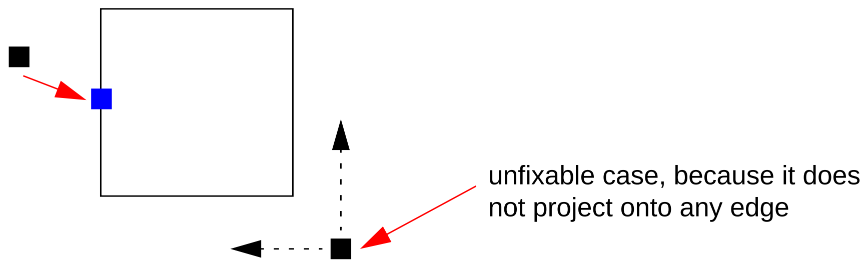
Description

This command goes through the spreadsheet file and either reports or fixes non-centered sample sites. An example of the results of a fix are shown in Figure 7-7. In this example there are two sites: the first is "fixed" by moving it to the center of the edge it projects onto. The second one cannot be fixed because it is unclear which edge the sample should be placed on. When the command encounters a sample sites that cannot be fixed, it reports this with an error message. Such unfixable sample sites should be examined in more detail by the user.

Example 1:

```
ssclean fix -ss sample.ss -m tp.gds -f setup.in
-->
contact  49 9   58497 620198 1   236.250 595.000   244.300   1.000
   No edge found near sample location.
contact  79 1   7316 801685 0   148.750 595.000   53.470   1.000
   Sample location not centered on edge.
```

Figure 7-7. Ssclean Attempts to Fix Off-center Sites



Usage

ssclean {**fix** | **report**} **-ss** ***input*** **-f** ***setup*** {**-m** ***gds*** | **-l** ***handle***} **-o** ***output***

Arguments

- **fix | report**

  Indicates whether the application should fix the bad data automatically, putting it into the output, or just report the bad data.

- **-ss *input***

  The input spreadsheet to be fixed or reported about.

- **-f *setup***

  The setup file.

- **-m *gds* | -l *handle***

  GDS test pattern file or a layout object handle (cannot use both).

- **-o *output***

  The output spreadsheet to create (only in “fix” mode).

sss_add

Tools Supported: Calibre WORKbench

Description

Appends a new sample set to an existing super spreadsheet object. A sample set consists of the spreadsheet filename, and values for the corresponding dose, defocus, and weight. This command returns the index of the newly appended sample set.

You can have up to a maximum of 128 spreadsheets in a single super spreadsheet object.

Note

See the section "Sample Points Spreadsheet" for information on spreadsheets, and the section "Super Spreadsheet File" for more information on super spreadsheets.

Usage

sss_add *sss_name* -ss *"filename"* -dose *dose_value* -defocus *def_value* -weight *wvalue*

Arguments

- ***sss_name***

  A required argument specifying a super spreadsheet object (previously created with sss_create).

- **-ss *"filename"***

  A required argument specifying the filename of a normal sample spreadsheet. The filename can be absolute or relative, and should be enclosed in quotes.

- **-dose *dose_value***

  A required argument specifying the dose used to create the spreadsheet data.

- **-defocus *defocus_value***

  A required argument specifying the defocus value used to create the spreadsheet data.

- **-weight *wvalue***

  A required argument specifying the weight given for the spreadsheet compared to another spreadsheet as part of the same super spreadsheet object.

Example

```
# Create sss tcl object from scratch and then
# append several regular ss files.
set OBJA [sss_create]
sss_add $OBJA -ss "standard_1.ss" -dose 1.1 -defocus 0.25 -weight 1.0
sss_add $OBJA -ss "standard_2.ss" -dose 1.2 -defocus 0.15 -weight 1.0
sss_add $OBJA -ss "standard_3.ss" -dose 1.0 -defocus 0.35 -weight 1.0
sss_add $OBJA -ss "standard_5.ss" -dose 1.1 -defocus 0.05 -weight 1.0
```

sss_cget

Tools Supported: Calibre WORKbench

Description

This command gets the value of the specified switch from the super spreadsheet object, for the sample set given by the specified *index* (-ss returns ss file name). If an *index* and switches are not specified, it returns all switch values of all existing sample sets.

Usage

sss_cget *sss_name* [*index* [-ss|-dose|-defocus|-weight]]

Arguments

- ***sss_name***

  A required argument specifying the handle for a super spreadsheet object.

- *index*

  An optional argument specifying the index number (the number of an individual spreadsheet within the super spreadsheet object, numbered starting with 0) to retrieve values for.

  If "end" is specified, it returns the specified parameters of the last sample set.

  If "all" or no value for *index* is specified, the command returns all the argument values of all the sample sets.

- -ss | -dose | -defocus | -weight

  An optional argument specifying which value to retrieve from the specified spreadsheet:

  - -ss — name of the spreadsheet file
  - -dose — dose value for this spreadsheet file
  - -defocus — defocus value for this spreadsheet file
  - -weight — weighting value specified for this spreadsheet file

  If *index* is specified, but no values are given, it returns all the parameter values for the specified spreadsheet object.

Examples

```
set obj [sss_create -in “sample.sss”]
#now get the dose value associated with the second sample set
sss_cget $obj 2 -dose
#get the values of all parameters from third sample set
sss_cget $obj 3
#get the values of all parameters of all sample sets
sss_cget $obj
```

sss_config

Tools Supported: Calibre WORKbench

Description

This command updates the configuration and contents of the super spreadsheet object for the sample set at the given index.

Usage

sss_config *sss_name index* [-ss "*filename*" | -dose *dose_value* | -defocus *defocus_value* | -weight *wvalue*]

Arguments

- ***sss_name***

  A required argument specifying the handle of a super spreadsheet object.

- ***index***

  A required argument, specifying the index of one of the sample spreadsheets within the super spreadsheet object. You can also specify one of the following keywords:

  - end — The last sample spreadsheet added to the object.
  - all — Apply the changes to all the sample spreadsheets in the object.

One of the following arguments must also be specified:

- -ss "*filename*"

  Reloads the contents of the sample spreadsheet object with the specified file. The filename should be enclosed in quotes.

- -dose *dose_value*

  Resets the dose for the sample spreadsheet object to the specified value.

- -defocus *defocus_value*

  Resets the defocus value used to create the sample spreadsheet data. Specified in microns.

- -weight *wvalue*

  Is the weight given for the spreadsheet compared to another spreadsheet in the same super spreadsheet object.

Examples

```
set obj [sss_create -in “/tmp/sample.sss”]
sss_config $obj 1 -dose 1.04
sss_config $obj end -defocus -0.14
sss_config $obj all -weight 1.0
```

sss_create

Tools Supported: Calibre WORKbench

Description

Creates a super spreadsheet Tcl object (sometimes referred to as an “sss object”). This command returns a handle which can be used later, similar to other Tcl variables.

Specifying the -in argument loads the super spreadsheet file matching the specified name. If the -in argument is not specified, then it will create an empty sss object.

Usage

sss_create [-in “*filename*”]

Arguments

- -in *filename*

  An optional argument specifying the pathname of a previously-created super spreadsheet file. The filename must be enclosed in quotes.

Example

```
# Create sss tcl object from scratch
set OBJA [sss_create]
# Create sss tcl object from input file
set OBJB [sss_create -in “sample.sss”]
```

sss_delete

Tools Supported: Calibre WORKbench

Description

This command deletes the sample set at the specified index from the given super spreadsheet object. Indices of the remaining sample sets will be reordered internally.

To delete the whole super spreadsheet object, use the Tcl command **unset**.

Usage

sss_delete ***sss_name index***

Arguments

- ***sss_name***

  A required argument, specifying the handle for a super spreadsheet object.

- ***index***

  A required argument, specifying the index of a spreadsheet object inside the super spreadsheet object. Specifying “end” instead of a numeric value deletes the last spreadsheet.

Example

```
sss_delete $OBJA end
sss_delete $OBJA 1
sss_delete $OBJA 2
# destroy the whole object
unset OBJA
```

sss_save

Tools Supported: Calibre WORKbench

Description

This command saves the contents of the given super spreadsheet object to the specified filename.

Usage

sss_save *sss_name* -out “***filename”***

Arguments

- ***sss_name***

  A required argument specifying the handle of a super spreadsheet object.

- **-out “*filename*”**

  A required argument specifying the pathname to write the super spreadsheet object to. The filename must be enclosed in quotes.

Example

```
sss_save $OBJA -out “sss_out_1.sss”
```

tepCompile

Tools Supported: Calibre WORKbench

Description

Generates a custom test pattern and associated spreadsheet file as specified by the given TPF file.

Usage

tepCompile ***tpffilename***

Arguments

- ***tpffilename***

  The name of the test pattern file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. For a complete description of the test pattern file format, refer to "Test Pattern File Format" on page 386.

Examples

Creates a test pattern from a specified TPF file:

Example 1: from a UNIX shell prompt:

```
# calibrewb -a tepCompile my_test_pattern.tpf
```

Example 2: from the Calibre WorkBENCH console prompt:

```
% tepCompile my_test_pattern.tpf
```

tepCreate

Tools Supported: Calibre WORKbench

Description

Generates a version 11 or 12 test pattern and associated spreadsheet file. Three pattern types are currently available: standard, contact, psm.

- To create an etch test pattern use the tepCreate etch command.
- To create a SEM calibration test pattern, use tepScal command.
- To create a verification test pattern, use the tepVerification command.
- To create a version 10 test pattern, use the tpcreate command.

Usage

tepCreate ***pattern*** [-version {11 | 12}] [-cdwidth *width*] [-cdspace *space*] [-gdsfile *gdsfile*] [-ssfile *ssfile*] [-markers *marker*] [-topcellname *cellname*] [-dbinnm *dbunit*] [-gridinnm *grid*]

Arguments

- *pattern*

  A required argument specifying the type of test pattern to generate. ***pattern*** must be one of {**standard | contact | psm**}.

  For a complete description of the V11 test pattern types, refer to "Test Pattern File Format" on page 386.

- -version { 11 | 12 }

  Specifies the version of the test pattern. Use version 12 for the standard type test pattern; for all other pattern types, use version 11.

- -cdwidth *width*

  An optional keyword/value pair defining the target CD for width, in nm. The default is 140.

- -cdspace *space*

  An optional keyword/value pair defining the target CD for space, in nm. The default is 140.

- -gdsfile *gdsfile*

  The name of the GDS output file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The default is system-generated, created by concatenating test pattern parameters: *pattern.width.space.dbunit.grid.gds*.

- -ssfile *ssfile*

  The name of the spreadsheet output file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The default is system-generated, created by concatenating test pattern parameters: *pattern.width.space.dbunit.grid.ss*.

- -markers *marker*

  An optional keyword/value pair indicating the type of marker pattern to include in the test pattern. *marker* must be one of {<u>none</u> **|** cross **|** oddeven **|** sem}. The default is none.

- -topcellname *cellname*

  An optional keyword/value pair specifying the name of the cell in the gds file, which contains the pattern structures. The default is TOP.

- -dbinnum *dbunit*

  An optional keyword/value pair defining the database unit, in nm, The default is 1.

- -gridinnum *grid*

  An optional keyword/value pair defining the grid unit, in nm, The default is 5.

tepCreate etch

Tools Supported: Calibre WORKbench

Description

Generates a Version 11 etch test pattern and associated spreadsheet file.

Usage

tepCreate *etch* [-minwidth *minwidth*] [-maxwidth *maxwidth*]
[-minspace *minspace*] [-maxspace *maxspace*] [-gdsfile *gdsfile*] [-ssfile *ssfile*]
[-markers *marker*] [-topcellname *cellname*] [-dbinnm *dbunit*] [-gridinnm *grid*]

Arguments

- -minwidth *minwidth*

  An optional keyword/value pair defining the target width, in nm. The default is 140.

- -maxwidth *maxwidth*

  An optional keyword/value pair defining the maximum target width, in nm. The default is 3000.

- -minspace *minspace*

  An optional keyword/value pair defining the target space, in nm. The default is 140.

- -maxspace *maxspace*

  An optional keyword/value pair defining the maximum target space, in nm. The default is 3000.

- -gdsfile *gdsfile*

  The name of the GDS output file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The default is system-generated, created by concatenating test pattern parameters: *pattern.width.space.dbunit.grid.gds*.

- -ssfile *ssfile*

  The name of the spreadsheet output file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The default is system-generated, created by concatenating test pattern parameters: *pattern.width.space.dbunit.grid.ss*.

- -markers *marker*

  An optional keyword/value pair indicating the type of marker pattern to include in the test pattern. *marker* must be one of {none | cross | oddeven | sem}. The default is none.

- -topcellname *cellname*

  An optional keyword/value pair specifying the name of the cell in the gds file, which contains the pattern structures. The default is TOP.

- -dbinnum *dbunit*

  An optional keyword/value pair defining the database unit, in nm, The default is 1.
- -gridinnum *grid*

  An optional keyword/value pair defining the grid unit, in nm, The default is 5.

Examples

A standard test pattern using the default values. The GDS and spreadsheet output files are *standard.140.140.1.5.gds* and *standard.140.140.1.5.ss*, respectively.

```
% tepCreate standard
```

tepScal

Tools Supported: Calibre WORKbench

Description

Generates a SEM Calibration pattern and associated spreadsheet file.

Usage

tepScal [-cdwidth *width*] [-cdspace *space*]
[-dbuinnm *dbunit*] [-gridinnm *grid*] [-topcellname *cellname*]
[-layoutsystem {GDS|OASIS}] [-gdsfile *gdsfile*] [-blocksize *sizenm*] [-border 0 | 1]
[-semmarkers {none | cross | oddeven | sem}]

Arguments

- -cdwidth *width*

  An optional keyword/value pair defining the target CD for line width, in nm. The default is 180.

- -cdspace *space*

  An optional keyword/value pair defining the target CD for space, in nm. The default is 180.

- -dbuinnum *dbunit*

  An optional keyword/value pair defining the database unit, in nm, The default is 1.

- -gridinnum *grid*

  An optional keyword/value pair defining the grid unit, in nm, The default is 5.

- -topcellname *cellname*

  An optional keyword/value pair specifying the name of the cell in the gds file, which contains the pattern structures. The default is TOP.

- -layoutsystem {GDS | OASIS }

  An optional argument specifying the output type. The default is OASIS.

- -gdsfile *gdsfile*

  The name of the output file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The default is semcal.oas.

- -border {0 | 1 }

  An optional switch which, if selected (specifying a 1) draws a border around the test pattern. The default value is 0 (no border).

- -semmarkers *marker*

  An optional keyword/value pair indicating the type of marker pattern to include in the test pattern. *marker* must be one of {none | cross | oddeven | sem}. The default is none.

tepVerification

Tools Supported: Calibre WORKbench

Description

Generates a verification test pattern and associated spreadsheet file based on the specifications supplied in the test pattern file *tpffilename*. If no filename is specified, it generates a test pattern using the default parameter values, six structures in each column, and the width and space varying from 0.180um up to 0.8um. The default parameter values are as follows:

- topcellname: TOP
- makers: none
- border: 0
- output files: *pattern.gds* + *pattern.ss*
- dbgrid: 0.001um
- designgrid: 0.001um

Usage

tepVerification [*tpffilename*]

Arguments

- *tpffilename*

  The name of the test pattern file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. For a complete description of the test pattern file format, refer to "Test Pattern File Format" on page 386.

Examples

Example 1: a verification test pattern using built-in default parameters.

```
% tepVerification
```

Example 2: a verification test pattern using parameters specified by TPF file *mytpf.tpf*.

```
% tepVerification mytpf.tpf
```

tpcreate

Tools Supported: Calibre WORKbench

Description

Generates a version 10 test pattern and associated spreadsheet file. Note that this is the older, version 10 test pattern, which does not contain GDS text objects, and so cannot be used with the Extract Sample function available in Model Center.

To generate version 11 or 12 test patterns, use the tepCreate command.

Usage

tpcreate *target filename* [*mode*]

Arguments

- ***target***

  The target CD for test pattern structures.

- ***filename***

  The name of the output GDS file, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application. The application also uses this string to generate the name of the spreadsheet file, by stripping away the file extension and appending the extension *.ss*.

- *mode*

  An optional argument specifying the type of test pattern to generate. Mode must be one of {extended | basic | contact | psm}. The default is extended.

  For a complete description of the V10 test pattern types, refer to the *Calibre WORKbench User's Manual*.

Examples

Example 1: a 180 nm target test pattern and associated spreadsheet file.

Note that the output is two files: *cell180.gds* and *cell180.ss*.

```
% tpcreate 180 cell180.gds
```

Example 2: a 180 nm test pattern without line end shortening hammerhead structures.

```
% tpcreate 180 cell180.gds basic
```

utp::table2sparse

Tools Supported: Calibre WORKbench

Description

This command converts empirical (measurement) data from a pitch array into the lookup_file format required for **utp::verify**.

Usage

utp::table2sparse ***table lookup_file***

Arguments

- ***table***

  The pathname for the file containing the lookup information in a dense format.

  This is a text database table using space- or tab- separated values, typically created in a spreadsheet program with data in one of the following formats:

  - **width/pitch** — widths in rows and pitches in columns. Cell 1,1 must have the word “width” in it. For example:

    ```
    #Comments are here
    width 200 300 400
    100 98 78 90
    150 120 111 115
    ```

  - **pitch/width** — pitches in rows and widths in columns. Cell 1,1 must have the word “pitch” in it. For example:

    ```
    #Comments are here
    pitch 100 150
    200 98 120
    300 78 111
    400 90 115
    ```

  - **space/width** — spaces in rows and widths in columns. Cell 1,1 must have the word “space” in it. For example:

    ```
    #Comments are here
    space 100 150
    50  120
    100 98
    150  111
    200 78
    250  115
    300 90
    ```

  In the input table, wafer CDs must be expressed in nm. The table can have empty cells. Be careful to put only *one* space (or tab) between numbers, because extra spaces (or tabs) will be interpreted as the empty cells.

- ***lookup_file***

  The pathname for the file to which the results of processing will be written. This file contains a lookup table in the form of a tab-separated database table. Each row of the table takes the form:

  ```
  pitch  mask_CDs  wafer_CD
  ```

Example

```
200  100 98
200 150 120
300  100 78
300 150 111
400  100 90
400 150 115
```

utp::verify

Tools Supported: Calibre WORKbench

Description

This command generates comparison data, comparing CDs resulting from PRINTimage simulations to CDs predicted by interpolating from the measurement data in a pitch lookup table. (It is assumed that OPC is performed based on the pitch structures, so the spreadsheet points to the pitch locations only.) To generate this data, it runs cdsample simulations for a corrected layout.

The command returns a five-column output table, with one row for each point in the sample spreadsheet. The table values are expressed in nm. The columns are:

1. pitch
2. OPC-corrected mask CD
3. target CD (width of the opc layer)
4. simulated wafer CD resulting from OPC-corrected mask
5. predicted wafer CD (predicted from the lookup table for width = OPC-corrected mask CD)

Usage

utp::verify **-clayer** ***correction_layer*** **-ss** ***spreadsheet*** **-lookup** ***lookup_file*** **-m** ***gds***
[-f *setup* -out *output_file*]

Arguments

- **-clayer *correction_layer***

  Required argument identifying the layer number for output layer from OPC. This is the corrected mask data.

- **-ss *spreadsheet***

  Required spreadsheet file. The command uses the values for the locations of pitches, x, y, and loc, and ignores all other information.

- **-lookup *lookup_file***

  Required path to a file containing the lookup CD information.

- **-m *gds***

  Required path to the GDS file containing the *correction_layer*.

- -f *setup*

  Optional path to the setup file. If not specified, the command uses the setup file that is currently loaded into the WORKbench application.

- -out *output_file*

  The path to a file to which the 5-column output from the command will be written.

Lookup File

The lookup_file is a text file containing measurement data from your test pattern. You can create this lookup file either manually using a text editor, or automatically using the utp::table2sparse command, described in the following section.

The lookup file format is as follows:

```
#Comments are here
p1 w1 cd1
p2 w2 cd2
...
pn wn cdn
```

where:

- p1...pn are pitches.
- w1...wn are mask CD widths.
- cd1...cdn are wafer CDs.

This information is used to predict the mask widths required to deliver the specified target wafer CDs. Prediction is based on interpolation of widths at a specific pitch. Pitches are *not* interpolated. Thus, the GDS file containing the features being corrected must sample only those pitches that can be found in the lookup table.

vsbIndex

Tools Supported: Calibre WORKbench, Calibre MDPView

Description

Builds a persistent index for a VSB chip. Calibre WORKbench uses the index in its rendering algorithm to increase the drawing speed at high altitudes.

The command adds a data line to the VSB chip configuration file (chip.cnf):

```
//!Calibre.Workbench.Index: “chip.idx”
```

If the data line already appears in the VSB chip configuration file, this command overwrites the previous version.

- If an index file is available, it will be loaded into memory and used during layout viewing unless the Minimum Object Size To Draw option is set to 0.
- If no index file is specified for a chip, or the one specified cannot be opened (or is out of date), Calibre MDPView works without an index file (and will issue a warning).

Usage

vsbIndex ***chipDirectory*** *indexFile*

Arguments

- ***chipDirectory*** — Is a required argument specifying the path to the VSB-11 or VSB-12 chip files.
- *indexFile* — Is an optional argument specifying the path name of the index file. It can be a relative or absolute path (paths that start with the forward slash character (/) are considered absolute). Relative paths are relative to the chip directory specified in the ***chipDirectory*** argument.

  The default value for this argument (if not specified) is “chip.idx,” which will be created inside the chip directory.

vt5gen

Tools Supported: Calibre WORKbench

Description

Generates a best-fit VT5 model, using an existing resist model, called the nominal model, as a template defining the model form (number of parameters, etc.) It is assumed that the template resist model and the resist model used create the samdata file have the same resist model options.

Note

Setting the environment variable WB_PRINT_MODEL causes Calibre WORKbench to write the eigenvalues to the file u.txt. This is for advanced users to use in possible principal component analysis.

The minimum eigenvalue criterion is printed in the terminal window during the model building step. This number is determined by finding the result of the smallest eigenvalue divided by the largest (SV0). You use the minimum eigenvalue criterion to exclude the next smallest eigenvector.

The minimum eigenvalue is computed when the calculated eigenvalue comes out smaller than 0.001 (in versions previous to 2006.1, the value used was 2*10^-4).

Usage

vt5gen -mode { **th** | **bias** | **offset**} **-data** ***samdata*** **-in** ***template*** **-out** ***vt5out*** **-ss** ***ssfile*** [-l *layout*] [-xcheck {*enableflag*}]

Arguments

- **-mode th** | **bias** | **offset**

  A required argument to specify the optimization mode.

  - In "th" mode it will fit the threshold polynomial.
  - In the "bias" mode it will fit the bias polynomial.
  - In the "offset" mode it will optimize the epeBias and threshold polynomial. This requires the use of the **-ss** *ssfile* and **-l** *layout* parameters.

- **-data** ***samdata***

  A required input file supplying the image shape parameters (IMAX, SLOPE, ISLOPE, IMIN, and FACTOR), densities (if the model requires), empirical and model thresholds, and measured and simulated EPEs. This file must be in the Sample Data File format. You can generate it using either cdsample or samlinev2.

- **-in *template***

  A required argument, specifying a template input model to use for choosing parameters for output model. If not set, vt5gen reads the VT5 from the setup file.

- **-out *vt5out***

  A required keyword/value pair defining the pathname for the model generated by this command.

- **-ss *ssfil***

  An optional keyword/value pair defining the pathname for a Sample Points Spreadsheet file. If present, the software uses then weights defined in this file give different optimization weightings to specific sample points.

- -xcheck {*enableflag*}

  An optional argument, specifies whether to run (*enableflag* = 1) or not to run (*enableflag* = 0) the cross-check procedure to determine the best min_eigenvalue criterion to build the model. The cross-check procedure uses a generalized cross-validation criterion (GCV), which is a one-out cross-validation, also known as N-1 fold validation.

  GCV is based on the philosophy that if an arbitrary measurement is left out, a model should predict this observation well. GCV optimization is also known as a “predictive sum of squares” (PRESS) in statistics theory. It can be written as:

  GCV(K(min_eigenval))=||T_{error}||/(m-k(min)eigenval))^^{2} ->min

  where:

  - ||T_{error}|| is a Euclidean norm of the modeling threshold error
  - m is the number of measurements
  - k is the number of retained eigenvectors (this number increases as min_eigenvalue decreases)

Example

```
% setenv WB_PRINT_MODEL 1
% vt5gen -data samdata.txt -in vt5in.mod -out vt5out.mod -ss sample.ss
```

If the resulting terms in vt5out.mod are as follows:

```
TPAR IMAX IMIN
ttermCount 4
TTERM 0.304233
TTERM -0.1 IMAX 1
TTERM 0.022 IMAX 2
TTERM -0.022 IMIN 1
```

then the columns of the matrix U, written to “u.txt” correspond to the coefficients in front of the modeling terms starting from the second term. (the first column of U corresponds to the IMAX term, the second column corresponds to the IMAX^2 term, and the third column corresponds to the IMIN term.

Chapter 8
WORKbench File Formats

Some of the Calibre WORKbench batch commands will:

- require input in a special format
- return results in a special format
- both

When you use the Calibre WORKbench GUI for performing simulations and creating models, you rarely need to know about these files. However, when you use the batch commands, you must be prepared to create, manipulate, or examine these files. This chapter provides a detailed description of each of the files you may need when using these batch commands.

Model File Format Descriptions

Model file format descriptions are described in the sections that follow.

Table 8-1. Model File Formats

Format
Optical Parameters File Format
Source Map File Format
Jones Pupil Data File
Sample Points Spreadsheet
Sample Data File
Super Spreadsheet File
VT5 Model File Description
Test Pattern File Format
Gauge Object Command Relationships

Optical Parameters File Format

The Optical Parameters file defines the parameters for an optical model.

Uses

Input for:

- opticsgen
- modelflow

Generated by:

- Calibre WORKbench Optical Model Tool

Other: Can be included inline in a setup file

Contents

This file must conform to the following syntax restrictions:

- Only one parameter per line.
- All parameter values must appear on the same line as the parameter name.
- Parameter names must begin in the first column of the line.
- The comment character '#' indicates that a row is a comment, in which case it can be unformatted text.

General Parameters

- **version *num***

  A required version number for the optical model. The default is 7 (8 is available, but not the default). Versions prior to 3 are not recommended due to minor interpolation inaccuracies within the model. Any versions prior to 4 can be read in and understood, and used *exactly* as they were in used in past releases. Any new model you create will be the most recent version by default, and any features present in earlier versions can be used with full backwards compatibility.

  - **version *3*** — This is our original model definition, in which the source is rotated by 90 degrees.
  - **version *4*** — For this version, the source is not rotated by 90 degrees. This is the only difference between version 3 and version 4. (While the source rotation is not a problem for 90-degree rotationally symmetric illumination sources, such as STANDARD, ANNULAR, and QUAD, rotationally asymmetric sources such as DIPOLE, require attention when choosing a version < 4.)
  - **version *5*** — For this version, the source is not rotated by 90 degrees. The differences between 4 and 5 are related to image normalization and where the image is calculated.

    - In Version 5, the image normalization is by the source area. The image can be calculated at any plane, regardless of whether or not it is inside the first film. This allows you to compute for models with TARC.
    - In Version 4, the image can ONLY be calculated inside the first film. The user has no control over image normalization, and the default image normalization depends on whether optical model is aerial or thin-film (in air vs. inside resist). For aerial models the default image normalization is by the source area. For thin-film models the default image normalization forces the open frame intensity to 1 at the first film interface.
  - **version *6*** — Starting with version 6, the decomposition of the TCC matrix into SOCS kernels utilizes a faster eigenvalue decomposition (EVD) routine, as opposed to a slower (but more general) singular value decomposition routine (SVD). The time required for EVD compared to SVD can be up to six times shorter, so the time for optical model creation can be improved significantly for large TCC matrices (more than ~100,000 elements). Note that there are a few examples where EVD fails to converge; in those cases, the algorithm reverts to SVD. The numerical differences of SOCS models generated with EVD or SVD are negligible.
  - **version *7*** — An updated version of version 6, which must be used for optical models that have kernel diffusion or non-TCCS kernels. Otherwise identical to version 6. This version is the default.
  - **version *8*** — The most current version of the optical model, which includes the **focusblur** primitives (and may include vector entrance pupil information in the 2006.4 release).

- **opticalsystem *lambda NA sigma*** [*sigma_in*]

  Required keyword and values defining the optical system parameters:

  - ***lambda*** — wavelength in microns (e.g., 0.248 or 0.193).
  - ***NA*** — numerical aperture setting. (NOTE: The numerical aperture can have a value of up to the refractive index of the ambient medium in the image space. The image space medium is assumed to be air (1) by default, so the NA value cannot exceed 1. In order to simulate NA>1 for immersion, you must specify the value of the ambient refractive index (using the **ambient** keyword)
  - ***sigma*** — partial coherence factor, assuming standard illumination.
  - *sigma_in* — for annular, Quasar, ASML_dipole, or CROSS_QUAD illumination only, defines the inner sigma. When this is specified, the sigma value represents the outer circle.

  Note that when using a COMPOSITE source, the ***sigma*** and ***sigma_in*** values specified here are ignored.

Illumination System Parameters

- **illumtype** { STANDARD | ANNULAR | QUAD | QUASAR | CROSS_QUAD | DIPOLE | SPLAT | OFFAXIS | ASML_DIPOLE | COMPOSITE }

  Required. Defines the type of source used. When the source type is COMPOSITE, you must specify the primitive shapes used to build the composite model, as described in "Syntax for Specifying a Composite Source" on page 355.

- illumspot *r1 r2*

  Defines an illumination spot with center at ***r1*** and a radius of ***r2***. (Default = no illum spots)

- illumangle *angle*

  Defines the illumination angle for either the QUASAR or ASML_DIPOLE sources. (Default = not set)

- **inline_sourcemap** *access_id* **{**

  sourcemap

  }

  Allows you to specify a source map file inside the optical file. Multiple source maps can potentially be used inside a single Optical file (there is no practical need for this yet). Moreover, you can combine inlined source maps with source maps defined in files. The *access_id* is a name you will later use to access the source map definition from the COMPOSITE source element (see also the "Syntax for Specifying a Composite Source" on page 355). The actual data of the inlined source map follow in the next lines and the format is the same as source maps defined in files (see "Source Map File Format" on page 364). The end of the source map data is signified by a line containing only a closing brace (}). Note that inlined source maps will significantly increase the size of the Optical file, so it is a good practice to define them at the end of the Optical file.

  The following limitations apply when using multiple inlined source maps:

  - All defined source maps in the Optical file have to be referenced (used) by some COMPOSITE source element.
  - The order by which the source maps are defined has to be the same as the order by which they are referenced through multiple elements of a COMPOSITE source.

- rotangle *rot*

  Defines the rotation angle with respect to the x-axis to apply to ASML_DIPOLE sources. (Default = 0)

- magnification *mag*

  Specifies the de-magnification factor (also known as the reduction factor), which is the ratio of object (mask) to image size (wafer). It is defined as the ratio of image to object numerical aperture. This factor activates the so-called *obliquity correction factor*, which accounts for the different sizes of entrance and exit pupils and arises from energy conservation considerations.[1]

NOTE: In the case of immersion lithography, the image NA can exceed 1. However, the object NA is always <1. In such cases, using the default value of de-magnification (1.0) will result in a user error. We recommend that you set the proper value of de-magnification (typically 4.0) when using vector, thin-film image models. (Default = 1.0)

- o Setting the reduction factor to larger than 1 is not recommended for scalar models. For a complete explanation, refer to "Reduction Factor", in Chapter 13 of the *Calibre WORKbench User's Manual*.
- o Note that in the Calibre WORKbench Optical Model tool, this parameter is shown in the Main tab as Reduction Factor.

Z-axis Parameters

- ambient *N*

  Defines the refractive index of the ambient medium. The value of N must be a positive number. The ambient is assumed to be above the film stack defined, and is assumed to fill the space between the exit pupil and the top of the first film. (Default: not required and assumed to be N=1 (vacuum) - required for immersion lithography simulation)

- **defocuslevels *num start step***

  Required. Defines the defocus information:

  - o ***num*** = number of planes.
  - o ***start*** = starting optical plane in microns. Negative values are distances measured in the direction toward the light source. Positive values are distance measured in the direction away from the light source. The origin of the coordinate system varies according to the type of model.
    - aerial imagery — the best focus in air.
    - thin-film imagery — the air/film interface.
  - o ***step*** = delta in microns between planes.
  - o For a complete explanation, refer to Selecting Image Planes Parameters for Aerial Image Models" and Selecting Image Planes Parameters for Thin-Film Models" in Chapter 13 of the *Calibre WORKbench User's Manual*.

- film *thickness N K "comment"*

  Defines a single film. N is the index of refraction. K is the absorption.[1] The value of K should be a negative number. The comment must be enclosed in quotes.

1. D.C. Cole, E. Barouch, U. Hollerbach, "Extending Scalar Aerial Image Calculations to Higher Numerical Aperatures," J. Vac. Sci. Technol. B. vol 10(6), pp. 3037-3041, Nov/Dec 1992
1. Another way to think of *N* and *K* is that they are the real and imaginary parts of the refractive index of the film.

When you specify multiple resist films, they are assumed to be stacked downward (in the direction of the propagating light) in the order specified in the file. This keyword is required when creating a vector model or thin film scalar model.

- beamfocus *zval*

  Required for thin-film models, defines the position into (or out of) the film stack at which the best focus plane in the ambient medium (air or water) would occur. A negative number indicates that the best focus in the ambient would have occurred outside of the film stack. A positive value indicates that the best focus in the ambient would have occurred somewhere within the film. The *zval* is given in microns relative to the ambient/first-film interface.

  Note that beam focus is NOT the location of best focus of the image formed inside the resist. The position of best focus in the presence of the resist film stack will in general be at z greater than 0 and can only be determined from the simulation. As a rule of thumb, the position of best focus will be at approximately n r * d when the imaginary part of the resist refractive index is small and the BARC effectively cancels reflections. n r is the real part of the resist refractive index and d is the distance between the top of the resist and the reference focal plane.

  When using beamfocus, the defocuslevels are understood as being relative to the air/first-film interface.

- substrate *N K*

  N and K are the real and imaginary part of the refractive index of the substrate. The value of K should be a negative number. The substrate is assumed to be underneath the film stack.This keyword is required when creating a vector model or thin film scalar model.

SOCS/Kernel Parameters

- approxorder *numkern*

  numkern defines the number of kernels in the (optics part of the) SOCS approximation to use. NOTE: In order to create and use a model for Hopkins-based batch simulations, you need to explicitly set approxorder to 0, since Hopkins does not use the SOCS approximation. <u>This is a required argument.</u>

- forcePower2LUT { 0 | 1 }

  Optional flag used to control the allowed relationship between the optical diameter and the kernel grid size. It has a direct effect on the hoodpix flag, as follows:

  - 1 = optical diameter must be a power of two multiple of the kernel grid units
  - 0 = optical diameter can be any integer multiple of the kernel grid

  NOTE: When this flag is set to 0, the lookup table calculation takes longer to complete. <u>(Default = 1)</u>

- hoodpix *diam*

  Defines the optical diameter in microns, which is the diameter of spatial influence of the optical system. If the forcePower2LUT flag is set to 1, this will be rounded up to the nearest power of two multiple of kernel grid units; otherwise, it will be rounded up to the nearest multiple of kernel grid units. This is a required argument.

- kerngrid *units*

  Defines the size of the kernel grid, in microns. This is also referred to as the optical grid size. This is a required argument.

- kernel *N* [TYPE *type*][SCALE *scale*] [SQUARE yes | no] [USESOCS yes | no]

  This keyword allows a function to be either convolved with, or to replace, the default SOCS kernel. There is an option for not squaring the kernel output value and choosing the scaling value of the kernel. The default is no kernel modifications, which uses unmodified SOCS kernels. Figure 8-1 provides a graphical explanation of the functionality provided by this keyword.

  - *N*

    An integer value specifying which kernel to replace. Allowed values are between 0 and N-1, where N is the number of optical model kernels as specified in the Number of Kernels field in the Optical Model Tool.

  - TYPE *type*

    When TYPE is omitted, the default SOCS kernel is used. The string comprising type defines the function which will either replace or be convolved with the default SOCS kernel. It can be either:

    tophat *out_radius in_radius*

    out_radius — outer radius of circular tophat, in microns.

    in_radius — inner radius of circular tophat in microns.

    gauss *sigma*

    sigma — sigma in microns of Gaussian.

  - SCALE *scale*

    Defines the scale on output of a kernel in the SOCS system. The default is to use the eigenvalue computed by the SOCS decomposition.

  - SQUARE yes | no

    Indicates whether to square the output of a kernel. Default = yes.

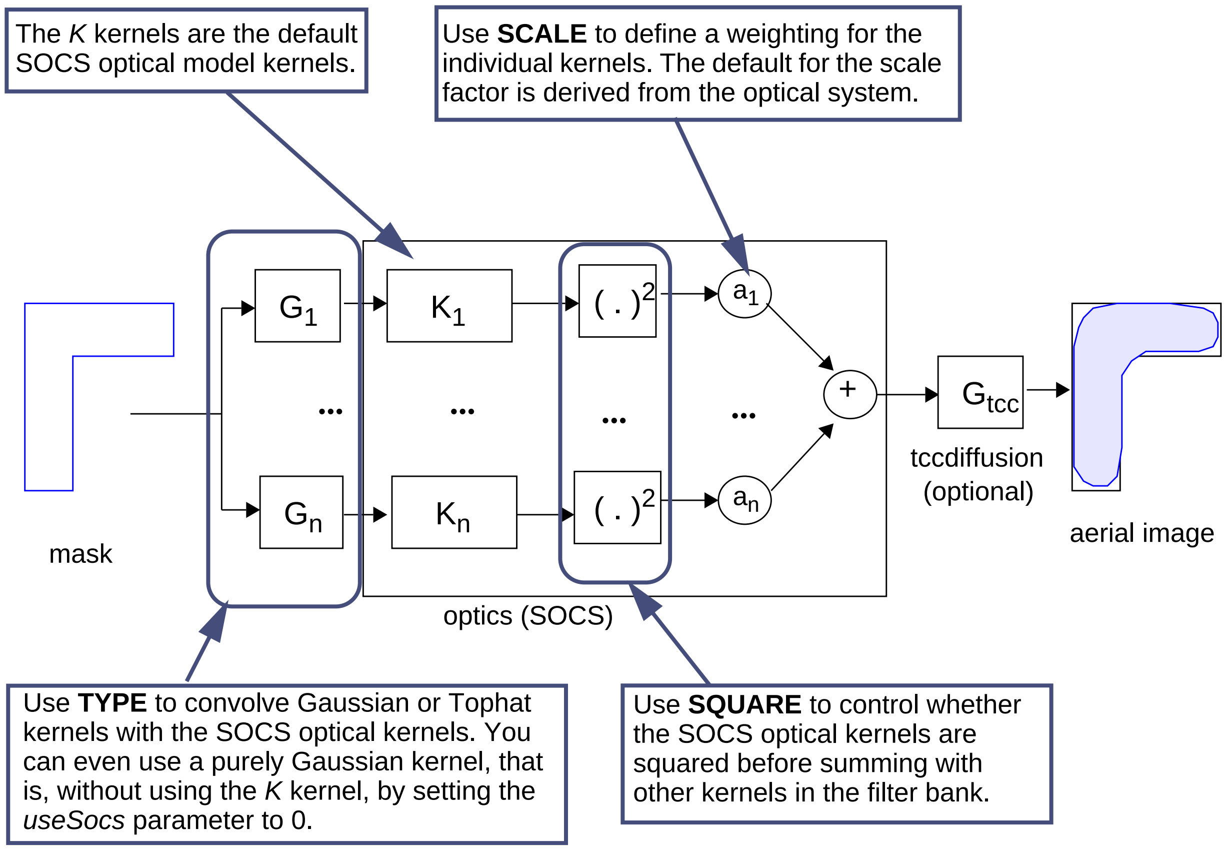
  - USESOCS yes | no

    Indicates whether or not the SOCS kernel is used. Allowed values are yes and no. Default = yes.

    yes — The SOCS kernel is used, convolved with the user-defined kernel, which will smooth it. The default is yes.

no — The SOCS kernel is replaced by the Gaussian kernel, using the extent defined by hoodpix.

Figure 8-1. Advanced Optical Functionality — kernel Keyword


The *K* kernels are the default SOCS optical model kernels.
Use **SCALE** to define a weighting for the individual kernels. The default for the scale factor is derived from the optical system.
G_1
K_1
$(.)^2$
a_1
...
+
G_{tcc}
tccdiffusion (optional)
aerial image
mask
G_n
K_n
a_n
optics (SOCS)
Use **TYPE** to convolve Gaussian or Tophat kernels with the SOCS optical kernels. You can even use a purely Gaussian kernel, that is, without using the *K* kernel, by setting the *useSocs* parameter to 0.
Use **SQUARE** to control whether the SOCS optical kernels are squared before summing with other kernels in the filter bank.

Pupil Parameters

- zernike *vals*

  Defines the 64 zernike coefficients, when they are used to define lens aberrations. The default is no aberrations.

- zernike_multiplier_flag {1 | 0}

  Optional flag used to define the method used for specifying Zernike coefficients:

  - 0 = as fractions of the wavelength
  - 1 = in microns

  (Default = 0)

- apodization_loss *a*

  Represents the lens apodization area loss present in the optical system. The default is 0. The pupil transmission function P(r) smoothly decreases from 1 to 0 in an apodized pupil. This can happen because of the imperfections of the lens coating or other reasons.

  At the extremes, the values of a are:

  - 0 = no loss, ideal pupil. (This is the default.)
  - 1 = lens is completely obscured, nothing gets through.

  For more information, refer to Accounting for Pupil Apodization Loss" in Chapter 13 of the *Calibre WORKbench User's Manual*.

Miscellaneous Parameters

- **engine {SPLAT | TCCcalc}**

  Specifies which TCC calculation engine to use. Default = SPLAT.

Note

NOTE: SPLAT is being phased out and is not allowed with a version setting >= 6, so **engine** is essentially a required parameter, since **TCCcalc** needs to be explicitly specified for versions higher than 5.

- focusblur [lorentz *gamma n*] [gauss *sigma*] [rectangle *width*] [integorder *num*]

  This setting enables modeling of three types of focus errors in the optics:

  1. focus errors because of chromatic errors of the laser source (finite source bandwidth)
  2. focus errors because of mechanical vibrations of the wafer stage along the optical axis
  3. focus errors because of systematic (purposeful) vertical motion or tilt of the wafer stage during the scan.

  The blurring of the image because of these three factors is modeled by computing a total blur distribution function, which comprises of three terms that respectively model these three sources of focus errors. The total image is a weighted average of partial images:

$$I_{blur}(x, y) = \frac{\int f_{blur}(z) I(x, y; z) dz}{\int f_{blur}(z) dz}$$

  The z variable of the unblurred image denotes either the location of the image plane for aerial image models (no films or substrate) or the beamfocus setting for thin-film image models.

  - The number of points for the numerical integration of the unblurred image is controlled by the optional parameter integorder. By default, integorder is 5, and the allowable range for *num* is: 3 <= *num* <= 51.

The total blur distribution is a convolution of up to three terms:

$$f_{blur}(z) = f_{BW}(z) \otimes f_{vib}(z) \otimes f_{tilt}(z)$$

$$f_{BW}(z) = \frac{\Gamma^n}{\Gamma^n + |2z|^n}$$

where both Γ and n are real numbers. $f_{BW}(z)$ is a modified Lorentzian function that models focus errors because of the finite laser bandwidth. The allowable ranges are: $0 <= \Gamma <= 0.5\mu m$ and $2.0 <= n <= 4.0$. You use the lorentz argument to specify these values. The default is for *n* is <u>2.8</u>.

$$f_{vib}(z) = \exp\left(-\frac{z^2}{2\sigma^2}\right)$$

$f_{vib}(z)$ is a Gaussian function that models focus errors that arise from vibrations of the wafer stage. The allowable range for σ is: $0 <= \sigma <= 0.5\mu m$. You use the gauss argument to specify these values.

$$f_{tilt}(z) = \mathrm{rect}\left(\frac{z}{W}\right)$$

$f_{tilt}(z)$ is a rectangle function that models focus effects that arise from tilting the wafer stage during the scan. The parameter W signifies the total focal range of each image point. The allowable range for W is: $0 <= W <= 1.0\mu m$. You use the rectangle argument to specify these values.

Note that all of the terms of the total blur distribution are optional, but at least one must be specified to perform the focusblur calculations.

<u>Examples</u>:

```
1) focusblur lorentz 0.1 2.8 gauss 0.05
```

Models focus errors because of finite laser BW and stage vibrations. The default integration order is used.

```
2) focusblur rectangle 0.2 integorder 3
```

Models a wafer that is tilted during the scan in a way that each image point goes through 0.2µm of focus around the center point. The integration order is dropped to 3.

- imagediffusion { <sigma> | weight1 <weight1> sigma1 <sigma1>
  weight2 <weight2> sigma2 <sigma2> }

<u>Default = 0 (no image diffusion).</u> Defines the Gaussian function(s) to convolve with the image.

Your options are:

- o Convolve the image with a Gaussian function of the indicated sigma. When this is present, the effective image is:

  $$I_{eff}(x, y) = \exp\left(\frac{-(x^2 + y^2)}{2 \cdot \sigma^2}\right) \otimes I(x, y)$$

- o Convolve the image with the weighted sum of two Gaussian functions. This option is available only when version is set to 5. When this is present, the effective image is:

  $$I_{eff}(x, y) = \left(w_1 \cdot \exp\left(\frac{-(x^2 + y^2)}{2 \cdot \sigma_1^2}\right) + w_2 \cdot \exp\left(\frac{-(x^2 + y^2)}{2 \cdot \sigma_2^2}\right)\right) \otimes I(x, y)$$

  The values for sigma are specified in µm and are restricted to be less than 0.2µm. Note that negative weights are not permitted with double Gaussian image diffusion.

- imagediffusion3d { 1 | 0 }

  The default is 0. An optional flag used to signal the model to compute the image diffusion in either 2D or 3D

  - o 1 = 3D
  - o 2 = 2D

  $$I_{eff}(x, y, z) = \exp\left(\frac{-(x^2 + y^2 + z^2)}{2 \cdot \sigma^2}\right) \otimes I(x, y, z)$$

  You must use this keyword in conjunction with imagediffusion, which defines the σ of the Gaussian function.

- image_normalization { source_area | integ_source_area | TCC0000 | first_interface | ambient_level | z_position *z_in_um* }

  Optional keyword, which allows you to normalize the image created by the optical model. Modifying the image normalization can affect TCCcalc output results.

  - Defaults (for versions < 5):
    - o Aerial models or, in general, models in a uniform ambient medium (scalar or vector) -**source_area** whenever possible, else **integ_source_area**
    - o Thin-film models (scalar or vector) - **first_interface**
  - Defaults (for versions >= 5): **source_area** whenever possible, else **integ_source_area**

  Note that image normalization is an advanced modeling feature, used primarily in research situations. For most production models, the default normalization is appropriate.

This setting relates to the image intensity of a transparent mask area that contains no geometry, sometimes referred to as "open frame intensity (OFI)". The allowed settings are:

- source_area — used for both aerial (ambient) models and thin-film models.

  For aerial image models, normalization by the source area results in OFI = 1 (This is not true if you use a reduction factor different than 1 or if you use apodization_loss). The source area is calculated analytically. Note that there are cases where analytical calculation of the source area is not possible, such as COMPOSITE sources that contain COHERENT elements or elements with MAP or ERF profiles.

  For thin-film image models, normalization by the source area will, in general, result in $OFI \neq 1$. Energy conservation is built-into the thin-film model, therefore variations in OFI with respect to the image plane location can be a result of energy loss from reflection in the air/resist interface or presence of standing waves inside the resist and other films.

- integ_source_area — used for both aerial (ambient) models and thin-film models.

  Calculates the source area through integration.

- TCC0000 — used for both aerial (ambient) models and thin-film models. Image normalization by TCC0000 makes OFI = 1 regardless of reduction factor, apodization_loss or image averaging over a number of planes. Image normalization by z_position z_in_um makes OFI = 1 at a plane specified by z_in_um.

- first_interface — used for thin-film models only. This is the default image normalization for thin-film models. This normalization results in OFI = 1. (In general, the open frame intensity will be different than 1 at other planes inside the resist film because of standing waves and/or attenuation).

- ambient_level — used for both aerial (ambient) and thin-film models.

  Image normalization by ambient_level makes OFI = 1 for aerial image models (or models in a uniform ambient medium), regardless of magnification or apodization loss. Using **ambient_level** in thin-film models will in general result in OFI < 1 inside the resist, because of energy lost from reflection, refraction and lossy media in the film stack.

- **z_position** *z_in_um* — used for thin-film models only.

  Image normalization by **z_position** *z_in_um* makes OFI = 1 at the plane specified by *z_in_um*.

- kerneldiffusion *sigma*

  This convolves all kernels with a Gaussian function of the specified sigma. This effectively performs a pre-convolution of the mask and is a short-cut for specifying the **kernel** keyword for each kernel.

  Using kerneldiffusion preserves the energy of the kernels, so the open field intensity is unchanged. The value of 0 for sigma is equivalent to no diffusion. The default is 0.

- model1D { 0 | 1 }

  An optional flag that specifies the generation of a one-dimensional optical model.

  Note that using this generated model is only valid with layout geometries that are strictly one-dimensional, such as iso lines or line/space patterns. Due to this serious limitation, 1D optical models have very specific uses, and are not designed for general purposes; you should not perform OPC based on 1D models. Examples where 1d optical models are useful include rapid stage 1 optimization in *modelflow* with 1d patterns, the workbench utility *od_analyze* and off-line testing and research work. Default = 0.

- pupilLookup *num*

  Optional keyword used to define the size of the internally generated pupil grid used when calculating the pupil function(s) and interpolating the pupil function(s) during the TCC calculations. When specified, the pupil grid is *num* x *num*. *num* must be an odd integer.

  This option is useful for speeding-up the calculation of the TCC matrix in the case of vector thin-film models, when many films (more than 1-2) comprise the film-stack. Recommended values for *num* are between 101 and 401.

  Default - no need to specify, the pupil function is internally calculated explicitly at every integration point.

- pupilPolarization { FILE *filename* | S | P }

  By default, the vector mode in TCCcalc models unpolarized pupil transfer functions.

  The electric field transformation from object to image field is given by the following equation:

$$\begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = \underline{P} \begin{bmatrix} U_x \\ U_y \end{bmatrix} \tag{1}$$

  where (U_x, U_y) is the electric field in the object (mask), (E_x, E_y, E_z) is the electric field in the image (wafer), and $\underline{P}$ is the complete transfer matrix that includes pupil and resist thin-film stack effects. The state of polarization when the field goes through the projection lens can in general be described by:

$$\begin{bmatrix} A_{s,\,out} \\ A_{p,\,out} \end{bmatrix} = \begin{bmatrix} J_{ss} & J_{sp} \\ J_{ps} & J_{pp} \end{bmatrix} \begin{bmatrix} A_{s,\,in} \\ A_{p,\,in} \end{bmatrix} \tag{2}$$

  or equivalently by:

$$\begin{bmatrix} A_{x,\,out} \\ A_{y,\,out} \end{bmatrix} = \begin{bmatrix} J_{xx} & J_{xy} \\ J_{yx} & J_{yy} \end{bmatrix} \begin{bmatrix} A_{x,\,in} \\ A_{y,\,in} \end{bmatrix} \tag{3}$$

where (A_s, A_p) or (A_x, A_y) is the electric field at the planar pupil-plane and the matrix $\underline{J}$ describes the polarization rotation at each point in the pupil.

The Jones matrix $\underline{J}$ is given either in s-p or x-y coordinate systems as shown by Equation (2) and Equation (3). Each point in the pupil requires a set of 8 numbers (the 4 elements of the $\underline{J}$ matrix are complex) to describe the rotation of the electric field polarization for the respective plane wave. (Recall that the coordinates of a point in the pupil correspond to direction cosines for the k-vector of the plane wave.)

In order to model the polarization transformations rigorously via the $\underline{J}$ matrix, 8 separate pupil functions need to be provided for the optical simulation. These pupil functions are also known as “Jones pupils” or “Jones pupil data.”

Specify one of the following arguments to the pupilPolarization parameter:

- o no argument (omit pupilPolarization)

  If the $\underline{J}$ matrix is the unitary matrix for all points in the pupil, it is evident that there is no polarization change when going through the lens. <u>This is the default setting</u>.

- o pupilPolarization S

  If $J_{ss} = 1$ and $J_{sp} = J_{ps} = J_{pp} = 0$ for all points in the pupil, the pupil acts as an s-polarization filter, by allowing complete transmission of the s-polarized plane waves and eliminating p-polarized waves. This can be simulated by specifying the S option.

- o pupilPolarization P

  If $J_{pp} = 1$ and $J_{ss} = J_{sp} = J_{ps} = 0$ for all points in the pupil, the pupil acts as a p-polarization filter, by allowing complete transmission of the p-polarized plane waves and eliminating s-polarized waves. This can be simulated by specifying the P option.

- o pupilPolarization FILE *path/filename*.jpf

  The general case of modeling pupil polarization requires a separate file that contains complete information about the Jones pupils. This functionality is similar to the source maps that model non-idealities in the illumination intensity. See the section “Jones Pupil Data File” for the specification of the Jones pupil file. A Jones pupil file can be included in the optical simulation by specifying the FILE option and a valid Jones pupil data file.

  The file *filename.jpf* needs to exist in *path*.

- skipModelHeader { 0 | 1 }

  When set to 1, this optional flag causes the application to hide the optical parameters used to generate the output optical model (note that this does not apply to inline models). The text file “Optical” and the source map file “illummap.smf” are not written in the output optical model directory. Also, all other binary files in the directory do not contain the usual text header with the optical information. This allows you to send generated optical model directories to other developers without revealing your optical settings.

Default = 0 - Write “Optical” and “illummap.smf” files in the output directory and write the text headers in each binary model file.

- TCCcalc_integorder *num*

  Specifies the number of integration points per maximum source sigma to use in the numerical integration routines of TCCcalc. This number should not be smaller than 8 or larger than 64. The default gives good accuracy for most practical optical modeling situations. Higher values than 16 are recommended in cases where high order aberrations are present or the illumination is specified through a file (MAP) and contains very rapid intensity variations. Default - no need to specify, 16 is used internally.

- tccdiffusion *sigma*

  Note: This keyword has been replaced by the keyword **imagediffusion**. **tccdiffusion** is supported for backward compatibility. The default is 0. Adds an aerial image diffusion term into the TCCs. This effectively performs a post-convolution of the aerial image with a Gaussian function of the indicated sigma. The value of 0 for sigma is equivalent to no diffusion.   When this is present, the effective aerial image is:

$$I_{eff}(x, y) = \exp\left(\frac{-(x^2 + y^2)}{2 \cdot \sigma^2}\right) \otimes I(x, y)$$

- splatline *line*

  The specified *line* is passed into SPLAT verbatim. Ignored for the TCCcalc models. The default is none.

- engine { SPLAT | TCCcalc }

  Specifies whether to use SPLAT or TCCcalc to create the model. The default is SPLAT.

- vectormodelflag { 1 | 0 }

  Tells whether to use the vector model. A value of 1 means create a vector model, 0 means create a scalar model. Note: When the vector model is used, the magnification value must be set to the correct reduction factor (typically 4). The default is 0.

Syntax for Specifying a Composite Source

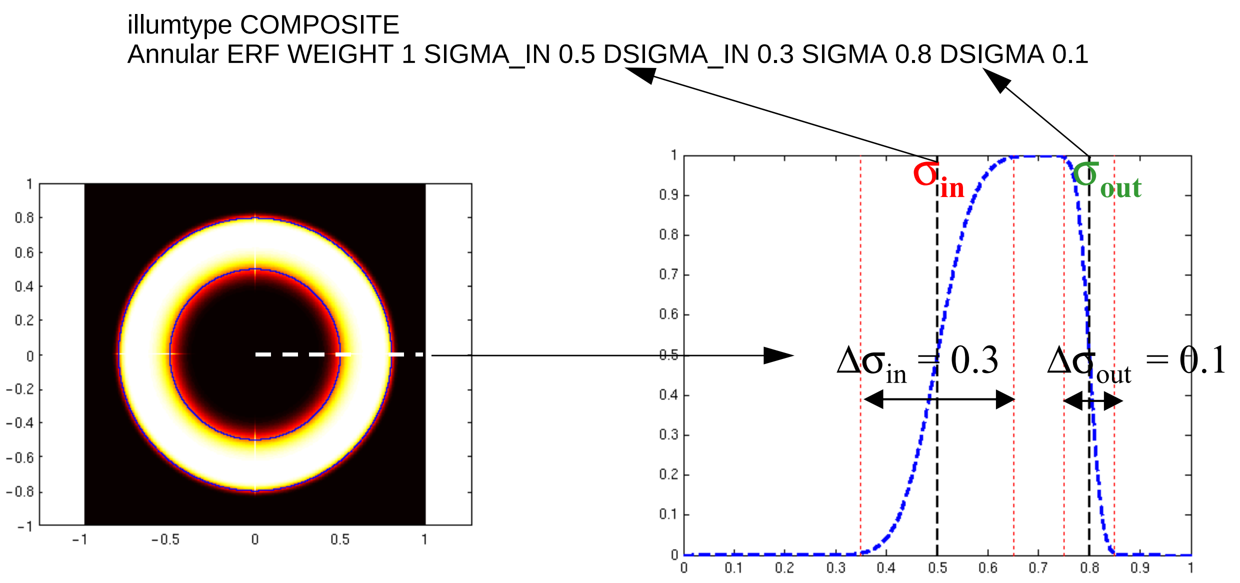
To use a composite source, you must build it up from primitive shapes. You begin by setting the illumtype to COMPOSITE:

```
opticalsystem lamda NA  <-- sigma is not needed here
illumtype COMPOSITE
```

You then include one line for each of the primitive shapes used to built the composite model. In addition to the primitive names and the parameters needed to define each, you must specify the profile and the weight:

- **Profile —** Describes the type of profile. Each primitive supports three profile types:
  - SHARP — the source is a step-function with abrupt transitions between bright and dark regions in the illumination. This is the most common.
  - MAP — the source is defined by a source map file used to show the exact value of the source intensity for each pixel in the source. When using a MAP profile, you must also include the argument FILE *map_file.* The actual file is expected to reside in the directory from which you invoked the Calibre WORKbench application.
  - ERF — the source is a function with a gradual transition between bright and dark regions that follows the error function profile raising from 0.5% to 99.5% within the user-defined delta range, as shown in Figure 8-2.

Figure 8-2. An ERF Profile for an Annular Illumination System



```
illumtype COMPOSITE
Annular ERF WEIGHT 1 SIGMA_IN 0.5 DSIGMA_IN 0.3 SIGMA 0.8 DSIGMA 0.1
```

- **Weight** — The weight of this primitive relative to the other primitives that comprise the source. Weights represent light intensity and can be negative or positive. The overall intensity at any given point is equal to the sum of all primitives at that point.

Syntax for Primitive Shapes Used with Composite Source

- STANDARD *profile* WEIGHT *w* SIGMA *sigma* {DSIGMA Δ*sigma*}[a] {FILE *map_file*}[b]{INLINE *id*}[c]
- ANNULAR *profile* WEIGHT *w* SIGMA *sigma* SIGMA_IN *sigma_in* {DSIGMA Δ*sigma* DSIGMA_IN Δ*sigma*_in}[a] {FILE *map_file*}[b] {INLINE *id*}[c]

- QUASAR *profile* WEIGHT *w* SIGMA *sigma* SIGMA_IN *sigma_in* ILLUMANGLE *angle* {DSIGMA Δ*sigma* DSIGMA_IN Δ*sigma*_in DILLUMANGLE Δ*angle*}[a] {FILE *map_file*}[b]{INLINE *id*}[c]
- QUAD *profile* WEIGHT *w* SIGMA_CNTR *sigma_center* ROTANGLE *rot_angle* RADIUS *radius* {DRADIUS Δ*radius*}[a] {FILE *map_file*}[b]{INLINE *id*}[c]
- CROSS_QUAD *profile* WEIGHT *w* SIGMA *sigma* SIGMA_IN *sigma_in* {FILE *map_file*}[b]{INLINE *id*}[c]
- DIPOLE *profile* WEIGHT *w* SIGMA_CNTR *sigma_center* ROTANGLE *rot_angle* RADIUS *radius* {DRADIUS Δ*radius*}[a] {FILE *map_file*}[b]{INLINE *id*}[c]
- ASML_DIPOLE *profile* WEIGHT *w* SIGMA *sigma* SIGMA_IN *sigma_in* ILLUMANGLE *angle* ROTANGLE *rot_angle* {DSIGMA Δ*sigma* DSIGMA_IN Δ*sigma*_in DILLUMANGLE Δ*angle*}[a] {FILE *map_file*}[b] {INLINE *id*}[c]
- OFFAXIS *profile* WEIGHT *w* SIGMA_CNTR *sigma_cntr*} ROTANGLE *rot_angle* RADIUS *radius* {DRADIUS Δ*radius*}[a] {FILE *map_file*}[b] {INLINE *id*}[c]
- COHERENT WEIGHT *weight* COHERENT_X *x* COHERENT_Y *y*

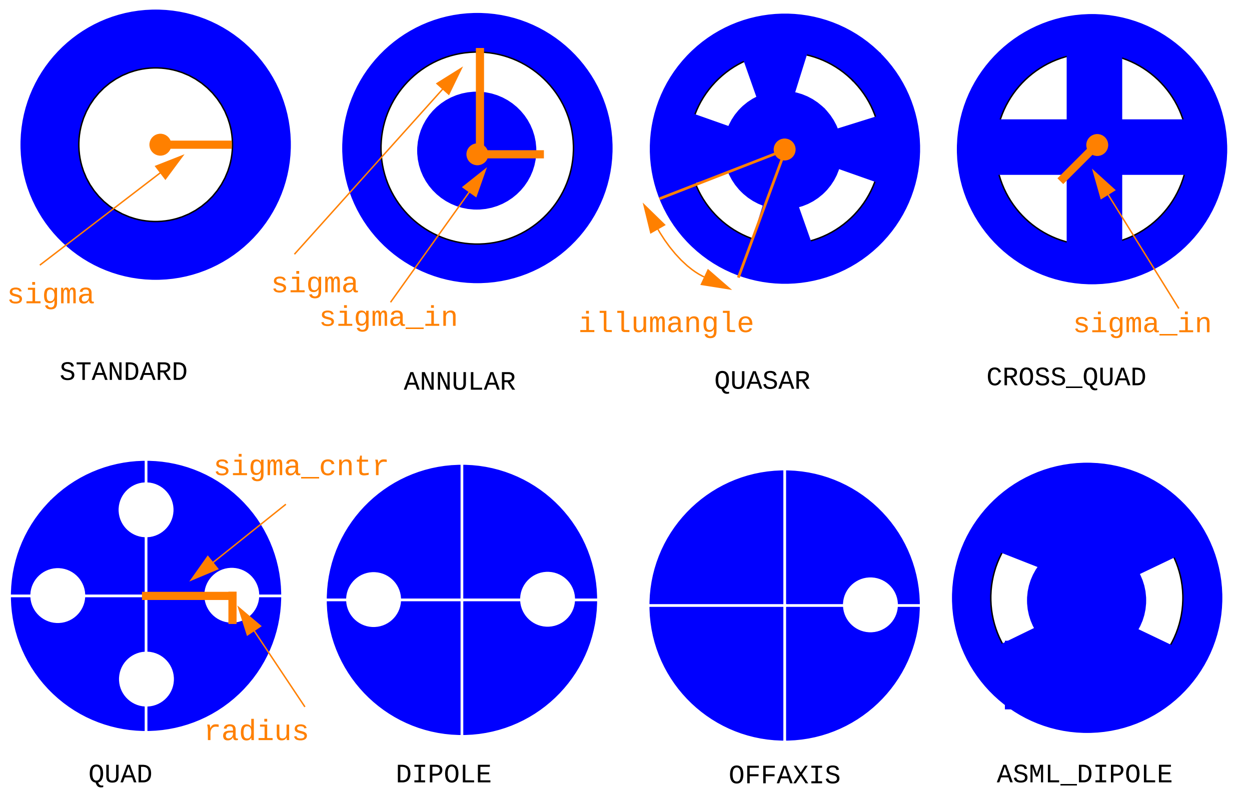
```
profile = {SHARP|ERF|MAP}
a Only valid for ERF profile
b Only valid for MAP profile
c Only valid for MAP profile, where id is the access identifier of the
inlined source map.
```

Note

The ERF profile is not implemented for CROSS_QUAD, DIPOLE, QUAD, and OFFAXIS.

A diagram showing the primitive types is shown in Figure 8-3, in which all parameters are defined. For source primitives which allow ROTANGLE, the drawings show the primitive with ROTANGLE=0.

Figure 8-3. Illumination Primitives



Simulating Source Polarization

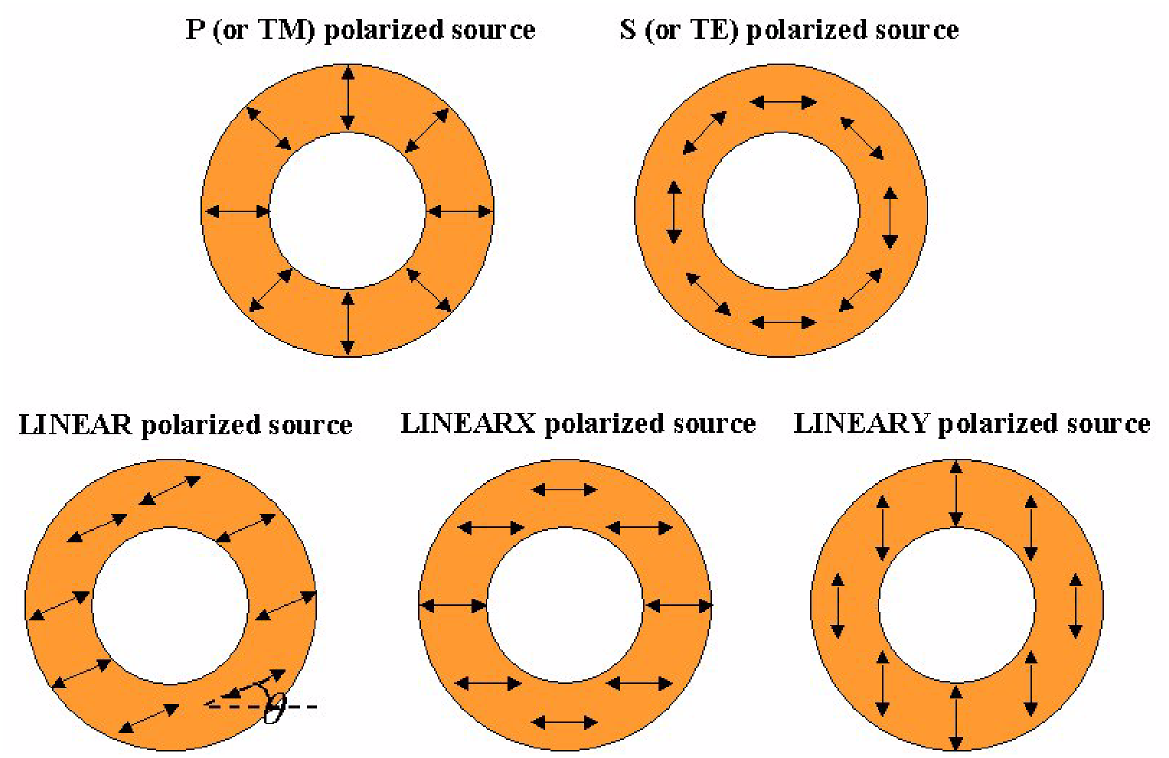
When using the vector model of TCCcalc it is possible to explicitly set the polarization properties of the illumination. This is useful for testing and also for simulating advanced source polarization mechanisms that are expected to be used in immersion lithography.

You can only set source polarization in COMPOSITE sources. There are six types that are supported:

- UNPOLARIZED
- LINEARX
- LINEARY
- LINEAR
- TE (transverse electric) or S (S-polarized)
- TM (transverse magnetic) or P (P-polarized)

The default source polarization in the vector model is UNPOLARIZED, which implies that the direction of oscillation of the electric field vector is random. Figure 8-4 shows the direction of oscillation of the electric field vector for the supported polarization types.

Figure 8-4. Different Types of Source Illumination Polarization



Each element of a COMPOSITE source can have up to three residual polarization types/weights. The specification of polarization follows the specification of all other relevant parameters of a COMPOSITE source element.

Syntax

```
<COMPOSITE source element with relevant parameters>
[ pol1 [ pol2 [ pol3 ]]]
```

where *pol1*, *pol2*, *pol3* is one of:

```
[ UNPOLARIZED w | TE w | TM w | LINEARX w | LINEARY w |
LINEAR w direction ]
```

- *w* is the weight of each residual polarization
- *direction* is a value in degrees relative to the x-axis. It only needs to be specified for LINEAR polarization.

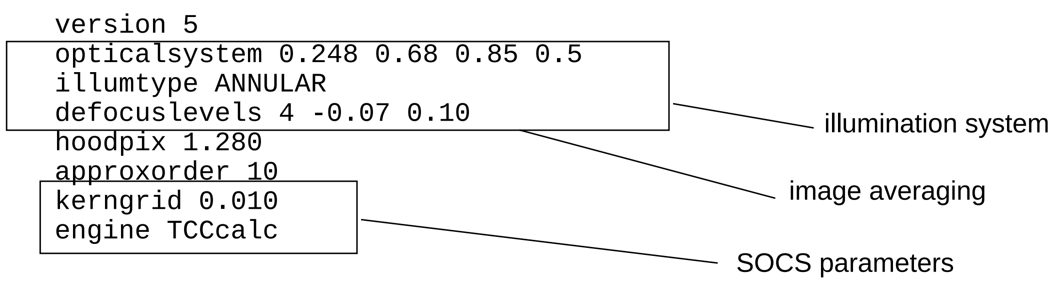
Examples:

```
1. 90% unpolarized light with 10% residual TE:


STANDARD ERF weight 1.0 sigma 0.6 dsigma 0.1 UNPOLARIZED 0.9 TE 0.1

2.100% unpolarized light - testing different ways of specification
STANDARD SHARP weight 1.0 sigma 0.6 UNPOLARIZED 1.0
STANDARD SHARP weight 1.0 sigma 0.6 TE 0.5 TM 0.5
STANDARD SHARP weight 1.0 sigma 0.6 LINEARX 0.5 LINEARY 0.5
STANDARD SHARP weight 1.0 sigma 0.6 LINEAR 0.5 45 LINEAR 0.5 135
STANDARD SHARP weight 1.0 sigma 0.6 LINEAR 0.5 20 LINEAR 0.5 110
STANDARD SHARP weight 1.0 sigma 0.6 UNPOLARIZED 0.2 S 0.4 P 0.4

3. 100% TE (or S) polarized QUASAR
QUASAR SHARP weight 1 sigma 0.75 sigma_in 0.5 illumangle 40 TE 1.0
```

Sample Optical Files

Example 1: A Simple Annular Model

Figure 8-5. Simple Annular Model Example

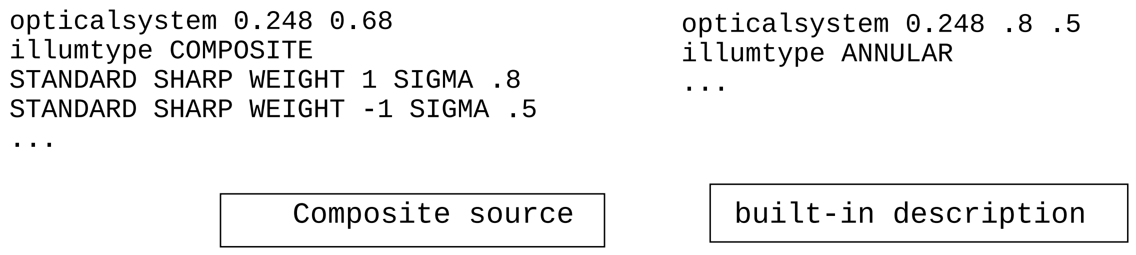


```
version 5
opticalsystem 0.248 0.68 0.85 0.5
illumtype ANNULAR
defocuslevels 4 -0.07 0.10
hoodpix 1.280
approxorder 10
kerngrid 0.010
engine TCCcalc
```

Example 2: Composing ANNULAR from STANDARD

Using the flexibility of the COMPOSITE source type, we can define an annular illuminator by combining two STANDARD sources. Of course, the ANNULAR source primitive should be used in practice, but as an examples of the flexibility of COMPOSITE source, this is illustrative.

Figure 8-6. Composite Source Example



```
opticalsystem 0.248 0.68
illumtype COMPOSITE
STANDARD SHARP WEIGHT 1 SIGMA .8
STANDARD SHARP WEIGHT -1 SIGMA .5
...
```

```
opticalsystem 0.248 .8 .5
illumtype ANNULAR
...
```

In this case, the choice of weights is very important. The weight of the larger STANDARD element is 1, and the smaller STANDARD element is -1 which subtracts out an inner circle.

Example 3: Using tccdiffusion

Figure 8-7 shows the effect of image diffusion on a standard optical model. The dashed line (red) is the non-diffused image and the solid (blue) line is the diffused image. You accomplish this by including an aerial image diffusion term in the TCC function by using the **tccdiffusion** keyword.This produces an effective aerial image:

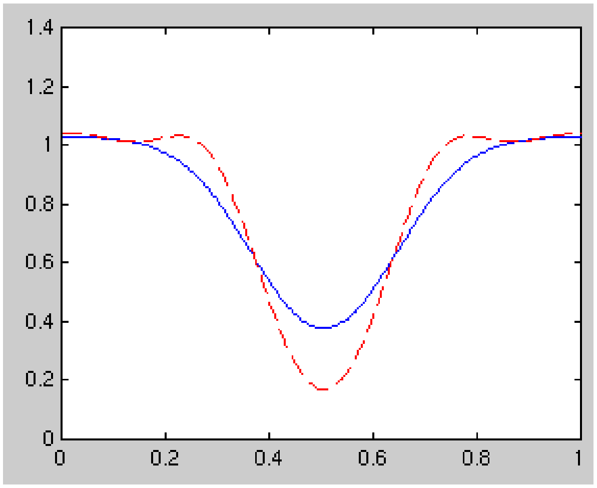
$$I_{eff}(x, y) = \exp\left(\frac{-(x^2 + y^2)}{2 \cdot \sigma^2}\right) \otimes I(x, y)$$

In this formulation, the value for sigma, σ, is set by the **tccdiffusion** keyword.

Figure 8-7. Image Diffusion on a Standard Optical Model

```
engine TCCcalc
opticalsystem 0.248 0.650 0.700
defocuslevels 1 0.000 0.050
magnification 4.
approxorder 10
hoodpix 1.280
kerngrid 0.010
illumtype STANDARD
tccdiffusion 0.065
```

diffusion of 65nm

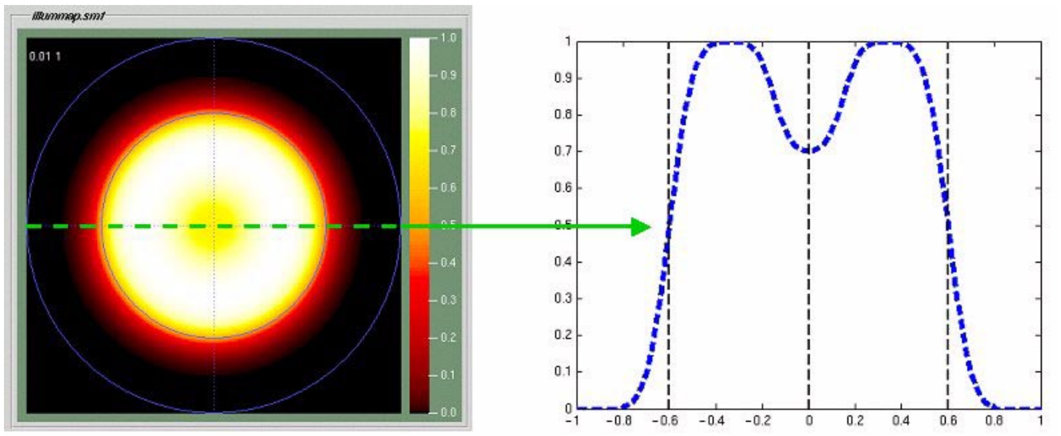


Example 4: STANDARD Source With Partial Central Obscuration (Using ERF Profile)

This example shows how to combine two STANDARD elements in the COMPOSITE source to create a STANDARD source with diminished intensity at the center (partial obscuration). Notice how the negative 0.3 weight of the second STANDARD element creates a 0.3 dip in the source intensity at s=0.

Figure 8-8. Using COMPOSITE Source ERF Profile

```
illumtype COMPOSITE
STANDARD ERF weight  1.0 sigma 0.6  dsigma 0.4
STANDARD ERF weight -0.3 sigma 0.15 dsigma 0.3
...
engine TCCcalc
```



Example 8: Mask writer model (bypass optics)

Using the ‘kernel’ keyword, it is possible to simulate a system which ONLY convolves the mask with a Gaussian kernel, without passing through an optical system. If the optics are bypassed with this command (USESOCS no), it still requires several dummy optical model file parameters to be set to make the format acceptable. In this case, the dummy “placeholder” information can be set to any value, and doesn’t affect the simulation. One or more kernels are sometimes used in a “maskwriter model”, although this example shows just a single kernel. To ensure that the optics are bypassed for all kernels, the ‘kernel’ keyword with “USESOCS no” must be set for each of the kernels.

Figure 8-9. Using ‘kernel’ to represent a maskwriter model.

```
version 4
opticalsystem 0.248 0.630 0.750
defocuslevels 1 0.050 0.020
approxorder 1
hoodpix 1.280
kerngrid 0.010
illumtype STANDARD
kernel 0 TYPE gauss 0.1000 SCALE 1 SQUARE no USESOCS no
```

Source Map File Format

The source map file is an ASCII text file (*.smf, *.src) that specifies the illumination intensity on a grid and can be used to create an optical model with the COMPOSITE source, using the MAP profile type. An output source map (named "*illummap.smf*") is also automatically generated in the optical model directory created by **opticsgen** whenever the TCCcalc engine is used. Note that you use this output file only for viewing purposes; it is never used during model creation. Therefore, you may notice differences between an input source map used for model creation and the output source map used for viewing. In particular, the input and output source maps may be defined on a different grid; the grid step size of the output source map is by default 0.01 (the user can change it by setting the environment variable SRC_MAP_STEP in .cshrc or .signaext) which can be different that the step size of the input source map.

Uses

Input for:

- bestsrcregion
- mergemap
- opticsgen — when creating a model with a composite source using MAP profile

Generated by:

- opticsgen
- Calibre WORKbench Optical Model Tool
- Third-party tools

Contents

```
[Version]
[Parameters]
step value
[DATA]
X Y intensity
...
```

Arguments

- [Version]

  An optional block at the top of the file specifying the version of the source map file. This section must begin with the keyword "[Version]". It ends with the next keyword.

- [Parameters] **step *value***

  A block at or near the top of the file used to define any relevant parameters. Of these, only **step** is required. This section must begin with the keyword "[Parameters]". It ends with the "[DATA]" keyword.

- [DATA] *X Y intensity* . . . .

  The main portion of the file, used to define the intensity of light at each pixel in the illumination source. This section must begin with the keyword “[DATA]”. It ends at the end of the file.

  This section contains a single line for each pixel for which the intensity is non-zero. (You need only specify non-zero pixels. Any pixel not defined is assumed to be zero.)

  The (X,Y) values for the pixels are specified in units of (step*N), where N is an integer index of the 2-D matrix of source map values.

  The pupil coordinates *X* and *Y* are unit-less, and represent normalized spatial frequencies. The *intensity* values of the illumination are also unit-less. They can have any non-negative value.

  - If multiplied by NA/lambda, they represent spatial frequencies in units of 1/um (if lambda is specified in microns).
  - If multiplied by NA, they represent direction cosines with respect to the x- and y-axes.

Example (Source Map File Format)

```
[Version]
[Parameters]
step 0.01
[DATA]
-0.0000 -0.8000 1
-0.1200 -0.7900 1
-0.1100 -0.7900 1
-0.1000 -0.7900 1
-0.0900 -0.7900 1
-0.0800 -0.7900 1
-0.0700 -0.7900 1
...
```

Jones Pupil Data File

Uses

Input For:

modelflow

Contents

A Jones pupil data file (*.jpf) contains complete information about the polarization properties of the projection lens. Refer also to the description of the keyword **pupilPolarization** in the section “Miscellaneous Parameters” for the “Optical Parameters File Format.”

The Jones pupil data file needs to be located in a path that is accessible during model generation.

Format

The format of a Jones pupil data file is text. Comments can exist on any line. A comment line is any line that begins with ‘%’, ‘!’, ‘#’ or ‘;’. A number of parameters and flags can be specified at the beginning of the file. Each parameter is specified in a separate line.

Parameters

- type { grid [*numx numy*] | polar | zernike | irregular }

  (Currently only the *grid* type is supported.) The Jones pupil data must be given on a rectangular regular grid. Points outside of the pupil may be omitted. In any case, the program ignores Jones pupil data for points outside of the pupil. The number of grid points numx and numy may be explicitly specified, but this is not mandatory, since the program can calculate the grid size and step from the data.

- format { real/imaginary | amplitude/phase }

  Format of the Jones pupil data. Default is *real/imaginary*

- wavelength *lambda_in_um*

  Default is *0.193*

- n_last_medium *float*

  Refractive index of the ambient medium of the last-lens-element. Default is *1.0* (vacuum)

- NA *float*

  The numerical aperture. Default is *1.0*

- mag *float*

  The magnification factor. A 4X reduction system corresponds to 0.25 magnification. Default is *0.25*

- field_point *xfloat yfloat*

  Field point (x,y) coordinates. Currently this value is not used by the program.

- Efield_map { XY | SP }

  Specifies whether the Jones matrix for each pupil point is given relative to the x-y or s-p coordinate system (see also Equation (2) and Equation (3)). Default is *XY*

- pupil_coord { normalized | dircos | NA }

  Specifies whether the coordinates of a pupil point are normalized spatial frequencies $(\hat{f}, \hat{g})$, direction cosines (s_x, s_y) of the respective k-vector, or relative to the numerical aperture NA. The following relations hold:

  $$s_x = \hat{f} \cdot \frac{NA}{n\_last\_medium}, \qquad s_y = \hat{g} \cdot \frac{NA}{n\_last\_medium}$$

  - If *normalized* pupil coordinates are used, the useful pupil is a unit circle with $\hat{f}$, $\hat{g}$ inside (-1, 1).

  - If *dircos* pupil coordinates are used, the useful pupil is a circle with s_x, s_y inside $\left(-\frac{NA}{n\_last\_medium}, \frac{NA}{n\_last\_medium}\right)$. If *NA* pupil coordinates are used the useful pupil is a circle inside (-NA,NA). Default is *normalized*

- DATA

  The string *DATA* must exist by itself on one line of the file. It signifies the beginning of the Jones pupil data, which must start immediately afterwards. An excerpt of a file would be:

```
...
DATA
px       py    jssa   jssp   jspa   jspp   jpsa   jpsp   jppa   jppp
-1        0     1      0      0      0      0      0      1      0
-0.99   -0.14   1      0      0      0      0      0      1      0
-0.99   -0.13   1      0      0      0      0      0      1      0
...
```

  The line "px py ..." is ignored by the parser and may be omitted. Its purpose is to label the columns of Jones pupil data. From the previous excerpt it can be deduced that the data is given in an amplitude/phase format and s-p pupil coordinate system. It should be stressed however that the format and coordinate system are specified by **format** and **Efield_map** keywords, so in the previous example the following 2 lines must exist in the beginning of the file, because these settings are different than the defaults:

```
format amp/phase
Efield_Map SP
```

Example 1

```
# this is a comment
% this is another comment
! and this is yet another comment
type grid 201 201
```

```
NA 0.5
pupil_coord normalized
format amp/phase
Efield_Map SP
DATA
px      py   jssa  jssp  jspa  jspp  jpsa  jpsp  jppa  jppp
-1       0     1     0     0     0     0     0     1     0
-0.99   -0.14  1     0     0     0     0     0     1     0
-0.99   -0.13  1     0     0     0     0     0     1     0
...
```

In this example, the format is amplitude/phase and the s-p coordinate system is used to specify the polarization transformations according to Equation (2). Because normalized pupil coordinates are used, non-zero pupil data can exist inside a unit circle only. The square grid has 201x201 points, which implies that the grid step is 0.01. Notice that the data appears to be representing an unpolarized pupil.

Example 2

```
type grid
NA 0.5
pupil_coord NA
format real/imaginary
DATA
px      py   jxxr  jxxi  jxyr  jxyi  jyxr  jyxi  jyyr  jyyi
...
-0.48   -0.14  1     0     0     0     0     0     0     0
-0.48   -0.13  1     0     0     0     0     0     0     0
-0.48   -0.12  1     0     0     0     0     0     0     0
-0.48   -0.11  1     0     0     0     0     0     0     0
-0.48   -0.1   1     0     0     0     0     0     0     0
-0.48   -0.09  1     0     0     0     0     0     0     0
...
```

In this example, the format is real/imaginary and the x-y coordinate system is used to specify the polarization transformations according to Equation (3). Because NA pupil coordinates are used, non-zero pupil data can exist inside a circle bounded by (-0.5, 0.5). The number of points of the square grid is not specified, but the step size can be deduced from the data to be 0.01. Notice that the data appears to be representing a pupil that acts as an x-polarization filter that allows only the x-component to go through unaltered while eliminating the y-polarized field.

Sample Points Spreadsheet

The sample points spreadsheet contains spreadsheet table data for the associated test pattern.

Uses

Input for:

- samlinev2
- cdruler
- cdsample
- modelfitness
- modelflow
- ss2gauge
- ssclean
- utp::verify
- vt5gen

Generated by:

- tepCompile
- tepCreate
- tepCreate etch
- tepVerification
- tpcreate

Contents

The sample points spreadsheet consists of spreadsheet table data, with each line in the file representing one row of spreadsheet data. That is, each row in of spreadsheet data is separated from the next by newline (return).

Comment lines are allowed, and must begin with the comment character '#'. Comment lines can be either unformatted text or a row of spreadsheet data that is currently disabled.

Each non-comment row has the following format:

name	row	col	X	Y	Loc	target	other	meas	weight

Spreadsheet Data

- **name**

  The name of the test structure (feature type). This can be any of the MGC structure types described in "Feature Types and Associated Subkeywords".

- **row**

  The row in which the test structure occurs in the test pattern (not the spreadsheet row). This is used by the Model Center Tool to plot data sequences.

- **col**

  The column in which the test structure occurs in the test pattern. This is used by the Model Center Tool to plot data sequences within a given row.

- **X**

  The x-coordinate of the sample point, represented in database units.

- **Y**

  The y-coordinate of the sample point, represented in database units.

- **loc**

  The sampling location. If this is 0, it indicates the measurement is to the inside of the polygon edge nearest the (x,y) coordinate. If this is 1, it indicates the measurement is to the outside of the polygon edge nearest the (x,y) coordinate.

- **target**

  The drawn dimension on the mask, at wafer scale.

- **other**

  A structure-specific value. For pitch lines, for example, this contains the space, so that (target+other) = pitch.

- **meas**

  The CD measurement for this structure.

- **weight**

  The relative weight of this structure for the purposes of model-fitting. Cannot be negative.

Example

A spreadsheet containing a single row of data, and the sample points spreadsheet file for that spreadsheet:

name	**row**	**col**	**X**	**Y**	**Loc**	**target**	**other**	**meas**	**weight**
iso	1	1	10500	10500	0	130	NA	109.5	1.0

```
# Sample points spreadsheet version 10

# ----------------------------------------
# name row col X Y Loc target other meas weight
iso 1 1 10500 10500 0 130 NA 109.5 1.0
# ----------------------------------------
```

Sample Data File

The sample data (*samdata*) file is a formatted text file consisting of lines of simulated and empirical measurements. This is an intermediate file used for batch model creation and model evaluation. These measurements correspond to the model fitness data generated by Model Center.

Uses

Input for:

- resistgenv2
- modelflow

Generated by:

- cdsample
- samlinev2

Other: Can be included inline in a setup file

Contents

With the exception of the first line, the file represents a table of data with one row for each structure in a test pattern. The file contains no comments. Each line is separated by newline (return).

The first line is a header, and defines the total number of rows and the number of columns in each row:

```
rows columns
```

All subsequent lines contain an array of numbers, each defining the value in the associated column of data.

The number of columns varies according to the number of CKERNEL convolutions in the model. Table 8-2 shows the parameter reported in each column. Note that for some parameters, the definition varies according to model type.

Table 8-2. Sample Data Parameters

Column	**Parameter**	**CTR, VTR or VTR-E**	**VT5**
1	Imax	first local maximum of intensity within the site	maximum intensity within a window of total width 2*searchRange (default searchRange = .5*lamda/NA)
2	Slope	maximum slope of intensity within site. If logslope option is on, it is the maximum slope of the log of the intensity	slope calculated at the reference threshold
3	Islope	intensity at the maximum slope	reference threshold
4	Imin	first local minimum of intensity within the site	the minimum value of the image within a window of total width 2*searchRange (default searchRange = .5*lamda/NA)
5	Factor	Factor calculated at ‘opc’ layer edge location. If slideTangentSites is on, calculated at the control point closest to location of maximum slope	Factor calculated at the reference threshold
(5+ i) for i= (1...n)	d_i	Not applicable	density[1]for Ckernel i
N-4	empThresh	actual threshold at which printing occurs (measured)	actual threshold at which printing occurs (measured)
N-3	modelThresh	**simulated threshold predicted by the model**	**simulated threshold predicted by the model**
N-2	empEPE	actual EPE (measured)	actual EPE (measured)
N-1	modelEPE	simulated EPE predicted by the model	simulated EPE predicted by the model formed by the thresh +bias polynomials
N	EPEv	modelEPE	simulated EPE predicted by the model formed by the thresh polynomial alone

1. Refer to the following page for a definition of “density”.

Where:

N = the number of columns
n = the number of CKERNELs

Density is defined as the area of the latent image of the structure normalized by the total kernel area, weighted according to the function specified for the kernel.

$$\mathrm{density}_n = f_n\left(\frac{\mathrm{Area}_{li}}{\mathrm{Area}_k}\right)$$

Where:

Area_{li} = area of the latent image for the structure, which is the simulated image generated using a CTR model with threshold = reference threshold.

Area_k = total kernel area

f_n = function specified for the kernel

n = index of the kernel

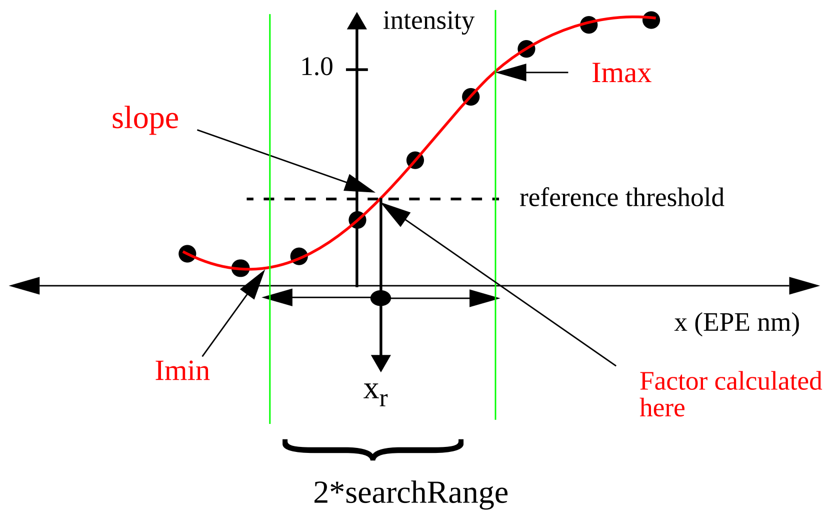
Example

```
2 9
0.97  5.1 0.52 0.22 0 0.25 0.28 -37 -30.6
0.96 5.2 0.50 0.19 0 0.29 0.28 -25.5 -28.0
```

More on VT5 Image Parameters

For VT5 models, the Imax and Imin are defined as the maximum value of the image within a window of total width 2*searchRange (default searchRange = .5*lamda/NA). The slope and curvature are calculated at the reference threshold.

Figure 8-10. Calculating Image Parameters for VT5 models



When the image does not pass through the referenceThreshold, the xr value is chosen to be the point on the cutline which is closest to the referenceThreshold, and all image parameters are derived using this value of xr.

Super Spreadsheet File

Super spreadsheet files (.sss) are used to calculate process windows by comparing multiple spreadsheets created at varying dose and focus settings. The super spreadsheet is a single file that is created by concatenating multiple spreadsheet files together. Each spreadsheet record within the super spreadsheet file is characterized by a unique dose and defocus setting (corresponding to the process conditions used to create the spreadsheet) as well as a weight value with which the specific spreadsheet data is to be taken into account during calibration.

Super spreadsheet files are used with the Process Window Calibration tool (see the *Calibre WORKbench User's Guide* for more information).

Usage

Input For:

- modelflow
- sss_add
- sss_cget
- sss_config
- sss_delete
- sss_save

Generated By:

- sss_create

Contents

Any line starting with a hash mark (#) is treated as a comment line.

The format of the .sss file is as follows:

```
# this is a comment
version version_number
sampleSetCount number_of_ss_files
sampleSet ss_number {
   dose dose_value1
   defocus defocus_value1
   weight weight1
   ssFile ssfilename1
      data {
         <lines of ss1 file>
      }
}
sampleSet ss_number {
   dose dose_value2
   defocus defocus_value2
   ssFile ssfilename2
   weight weight2
```

```
            data {
               <lines of ss2 file>
            }}
```

where:

- *version_number* — Is the version number of the super spreadsheet file. Currently this is always 1.
- *number_of_ss_files* — Is the count of the included .ss files. There is a hard limit of 128 spreadsheet files allowed per super spreadsheet file.
- *ss_number* — Is the numbered index of the spreadsheet. This is a sequentially increasing number from 1 to *number_of_ss_files.*
- *dose_value* – Is the dose value for this spreadsheet.
- *defocus_value* – Is the defocus value for this spreadsheet.
- *weight* – Is the weight given for this spreadsheet file. All the rows of this instance of the spreadsheet file are effectively multiplied by this value. If the weight specified is negative, this .ss file is skipped during simulations.
- *ssfilename* – Is the name of the original .ss file. Maintained only for GUI reference, and does not affect optimization results.

Example Super Spreadsheet Commands

The following example code highlights the creation of a super spreadsheet using the various super spreadsheet commands:

```
# Create sss tcl object from scratch and then
# append several regular ss files.
set OBJA [sss_create]
sss_add $OBJA -ss standard_1.ss -dose 1.1 -defocus 0.25 -weight 1.0
sss_add $OBJA -ss standard_2.ss -dose 1.2 -defocus 0.15 -weight 1.0
sss_add $OBJA -ss standard_3.ss -dose 1.0 -defocus 0.35 -weight 1.0
sss_add $OBJA -ss standard_5.ss -dose 1.1 -defocus 0.05 -weight 1.0
# Save it to an sss file.
sss_save $OBJA -out sss_out_1.sss
# Delete some ss from the sss object, record the states and at the end save 
the last
# state to another sss file.
sss_delete $OBJA end
sss_delete $OBJA 1
sss_delete $OBJA 2
sss_save $OBJA -out sss_out_2.sss
# Create sss object by reading in an sss file and save it to another file.
set OBJB [sss_create -in sss_out_1.sss]
sss_save $OBJB -out sss_out_3.sss
# This tests an asignment of variable.
set OBJC $OBJB

# Just for debugging.
puts $OBJA
puts $OBJB
puts $OBJC
# Delete objects.
unset OBJA
unset OBJB
# Save result of an asignment to a file.
sss_save $OBJC -out sss_out_4.sss

# Destroy the object.
unset OBJC
```

VT5 Model File Description

Creating a VT5 model results in a .mod file that fully defines that model. This file is the model itself, which you can include inline in a setup file or reference using the **resistpolyfile** keyword in a setup file or the **vt5** parameter in Model Flow.

Uses

Input for:

- vt5gen (to serve as the nominal (initial) model
- modelflow (to serve as the nominal (initial) model

Generated by:

- vt5gen
- modelflow

Other: Can be included inline in a setup file

Definitions

```
EPEf := final EPE = EPEv + bias
EPEv := EPE predicted by VTR at threshold value Tv
Tv   := VTR threshold = Pt(imax,slope,factor,imin,islope,d1...dn)
bias := EPE bias =Pb(imax,slope,factor,imin,islope,d1...dn)
Pb   := bias polynomial expression
Pt   := threshold polynomial expression
EPEr := reference threshold EPE
Tr   := reference threshold
```

Contents

The VT5 model consists of statement lines, blank lines, and comment lines. Each statement must be on a line by itself. Comment lines must begin with '#'.

General Statements

- version *num*

  The model version number. Currently, the only VT5 version supported is 3.

- type VT-5

  This indicates the type of model to be required. At present, only VT5 models are supported.

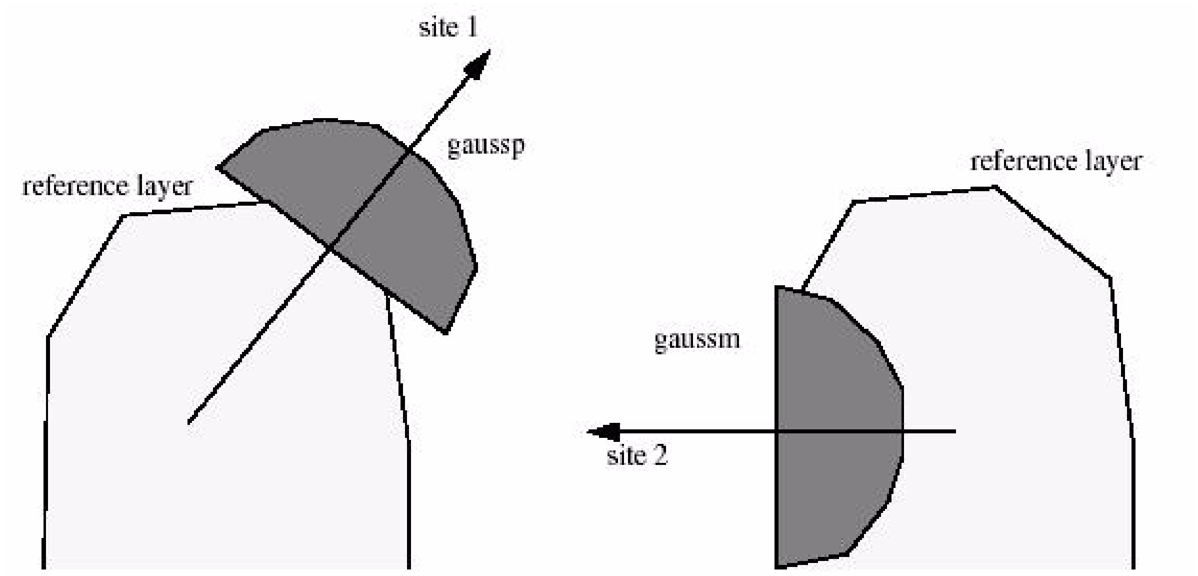
Density Kernel Definition Statements

- CKERNEL | DKERNEL *which expression*

  A description of the convolution kernel with which the reference aerial image will be convolved. You must supply one CKERNEL statement for every convolution. Since the convolution kernel is flipped, CKERNEL can only be used for symmetrical kernels (gauss); for the asymetrical kernels (gaussm and gaussp) use DKERNEL instead.

Use DKERNEL statements to represent the directional density kernels **gaussm** and **gaussp**. Directional density kernels are aligned in the X-axis with the VT5 site, with the center of the kernel positioned at the intersection of the reference layer and the site:

Figure 8-11. gaussp and gaussm Examples



In the gaussp example, the density value for site 1 is 0, since the kernel does not intersect the reference layer. In site 2, the density value is almost 1, because the kernel is almost completely covered by the reference layer.

- *which* — The name of the kernel. Names take the form Dn where D stands for “Density” and n represents the index of the kernel, such as D1, D2, and so on. You can use CKERNELs as terms within bias polynomials or threshold polynomials. When you do, you refer to them by these names.
- *expression* — A Tcl expression that gives the value of the kernel at a each coordinate. The expression can be any function of (x,y,r) in microns, such as:

  ```
  kernel = f(x,y,r)
  ```

  You can also take advantage of built-in expression shortcuts:

  gauss *sigma* - A symmetrical Gaussian kernel (used with CKERNEL) with user-specified sigma (microns). This is a shortcut for a Gaussian kernel:

$$K(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2 + y^2}{2\sigma^2}}$$

gaussm *sigma* - A half-gaussian directional density kernel (used with DKERNEL) with a user-specified sigma used for minus gaussian kernels:

$$K(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2 + y^2}{2\sigma^2}}, x \leq 0 \qquad 0, x > 0$$

gaussp *sigma* - A half-gaussian directional density kernel (used with DKERNEL) with a user-specified sigma used for positive gaussian kernels:

$$K(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2 + y^2}{2\sigma^2}}, x \geq 0 \qquad 0, x < 0$$

tophat *outer inner* - A tophat kernel which whose value is 1 when $\text{outer} > r \geq \text{inner}$ and 0 otherwise (inner and outer specified in microns).

You define the diameter and gridsize for the kernels through the model parameters **hoodpix** and **kerngrid**, respectively.

- hoodpix *diam*

  The interaction diameter of all CKERNELs. This value must be a multiple of the database units. Note that this is not the same value as the hoodpix for an optical model (also called optical diameter).

- kerngrid *units*

  The kernel gridsize, specified in microns.

- densityLayer {refimage | mask | target}

  An optional statement used to specify the layer to be convolved with the density kernels. The default value is **refimage**, which refers to the reference aerial image.

Threshold Polynomial Definition Statements

- TPAR {IMAX | SLOPE | FACTOR | ISLOPE | IMIN | D1 | ... | D16}*

  An optional statement used to indicate that the model is defined through a threshold polynomial. The arguments to this statement declare the parameters to be used in the threshold polynomial. You can supply as many TPAR statements as needed.

  In the .mod file, the **TPAR** statement must appear before any **TTERM** statements that define the actual terms and their coefficients.

- ttermCount *cnt*

  This statement is required when using TPAR. It defines how many TTERMs will be in the model. If the number of TTERM statements does not match this number, it will be a user error.

- TTERM *coef* [*VALUE POWER*]*

  Required when using TPAR, each TTERM statement defines a single term in the polynomial. The first value, *coef*, defines the scalar coefficient for this polynomial term. The VALUE POWER list defines the parameter(s) that serve as values for the term and the exponents for those values. The VALUE must be one of {**IMAX | SLOPE | FACTOR | ISLOPE | IMIN | D0 | ... | Dn**}. The list can contain up to two values.

  For example, the following statement describes a single monomial term:

$$\mathrm{coef} \bullet \prod_{i} (\mathrm{VALUE}_i^{\mathrm{POWER}_i})$$

  of the threshold polynomial:

$$T_v = \sum \mathrm{coef} \bullet \prod_{i} \mathrm{VALUE}_i^{\mathrm{POWERi}}$$

- minThreshold *min*

  The lower limit on the output threshold from the threshold polynomial.

- maxThreshold *max*

  The upper limit on the output threshold from the threshold polynomial.

Bias Polynomial Definition Statements

- BPAR {IMAX | SLOPE | FACTOR | ISLOPE | IMIN | D1 | ... | D16}*

  An optional statement used to indicate that the model is defined through a bias polynomial. The arguments to this statement declare the parameters to be used in the bias polynomial. Multiple parameters are allowed.

In the .mod file, the **BPAR** statement must appear before any **BTERM** statements that define the actual terms and their coefficients.

- btermCount *cnt*

  This statement is required when using BPAR. It defines how many BTERMs will be in the model. If the number of BTERM statements does not match this number, it will be a user error.

- BTERM *coef* [*VALUE POWER*]*

  Required when using BPAR, each BTERM statement defines a single term in the polynomial. The first value, *coef*, defines the scalar coefficient for this polynomial term. The VALUE POWER list defines the parameter(s) that serve as values for the term and the exponents for those values. The VALUE must be one of {**IMAX | SLOPE | FACTOR | ISLOPE | IMIN | D0 | ... | Dn**}. The list can contain up to two values.

  For example, the following statement describes a single monomial term:

$$\mathrm{coef} \bullet \prod_{i} (\mathrm{VALUE}_i^{\mathrm{POWER}_i})$$

  of the threshold polynomial:

$$T_v = \sum \mathrm{coef} \bullet \prod_{i} \mathrm{VALUE}_i^{\mathrm{POWERi}}$$

- maxBias *max*

  The upper limit on the absolute value of the output bias from the bias polynomial.

Miscellaneous Statements

- bound *param lo hi*

  An optional statement that defines upper and lower bounds for TPAR or BPAR parameters. During evaluation of the polynomial, all bounded inputs that are outside of the bounds will be limited to the bounds. You must supply one **bound** statement for every parameter to be bounded.

- referenceThreshold Tr

  A required statement that defines the threshold value used to compute the following:

  a. The slope — The slope for any site is calculated at the point where the intensity equals the reference threshold. If no such point exists, the maximum slope is used.

  b. The reference image — The reference image, which is convolved with the CKERNELS, is calculated using a constant threshold equal to this value.

- sampleSpacing *spacing*

  A required statement defining the spacing between control points on the sites.

- minEigenval *val*

  A required statement defining the lowest eigenvalue to use in linear least square fitting that occurs during model building. The *val* must be > 0.001 (previous to Calibre 2006.1, this value must be > $2*10^{-4}$).

- modelFile *fname*

  The name of this model.

- epeBias *val*

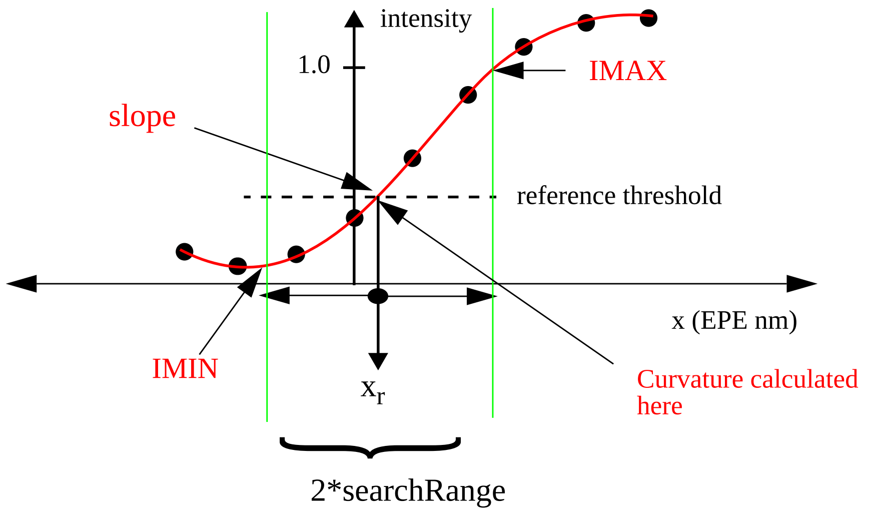
  A constant bias (fixed shift) to apply to all EPEs resulting from a simulation.

- opticalModelBinding *modelname*

  An optional statement supplying the name of an optical model. If defined, then this VT5 model will only work with the indicated optical model.

- searchRange *distance*

  An optional statement that defines an optional distance to search from the reference location when finding the IMAX and IMIN values. It searches along the image over an entire distance of 2*searchRange, meaning that it searches to the left and to the right by searchRange. If this value is not set (or set to a negative number) the application defaults to 0.5*lamda/NA.



- centerRegression {0 | 1}

  An optional switch that indicates that the design matrix was centered before performing regression. This allows building models with a larger first coefficient, making them closer to a CTR model, and therefore, more stable.

Example 1: CTR model

```
type VT-5
ttermCount 1
TTERM 0.2
```

Example 2: VTR model

```
type VT-5
TPAR IMAX SLOPE
ttermCount 6
TTERM 0.2
TTERM 0.073 SLOPE 1
TTERM 0.383 IMAX 1
TTERM -0.09 IMAX 2
TTERM 0.103 IMAX 1 SLOPE 1
TTERM 0.032 IMAX 2 SLOPE 1
maxThreshold .3
minThreshold .2
sampleSpacing .020
```

Example 3: VTRE model

```
type VT-5
TPAR IMAX SLOPE FACTOR
ttermCount 8
TTERM 0.2
TTERM 0.073 SLOPE 1
TTERM 0.383 IMAX 1
TTERM -0.09 IMAX 2
TTERM 0.103 IMAX 1 SLOPE 1
TTERM 0.032 IMAX 2 SLOPE 1
TTERM 0.120 FACTOR 1
TTERM -0.09 FACTOR 2

maxThreshold .3
minThreshold .2
sampleSpacing .020
```

Example 4: Double gaussian model

```
type VT-5

referenceThreshold 0.2
hoodpix 1.28
kerngrid 0.01
CKERNEL D1 "gauss 0.2"
CKERNEL D2 "gauss 1.7"

BPAR SLOPE D1 D2
btermCount 3
BTERM 0.073 SLOPE 1
BTERM 0.09 D1 1
BTERM -0.027 D2 1
maxBias 0.080
```

Example 5: Directional Density

```
version 3
type VT-5

referenceThreshold 0.2
hoodpix 1.28
```

```
kerngrid 0.01
sampleSpacing .020

DKERNEL D1 “gaussm 0.2”
CKERNEL D2 “gauss 1.7”

TPAR D1 D2
ttermCount 3
TTERM 0.2
TTERM -0.01 D1 1
TTERM 0.021 D2 1
```

Test Pattern File Format

A text format for generating verification test patterns using the tepVerification command and custom test patterns using the tepCompile command.

Suggested file extension for files containing text that conforms to this format are **.tpf*, and **.TPF*.

Uses

Input for:

- tepVerification
- tepCompile

Generated by:

- Must be generated "by hand".

Contents

A test pattern file must conform to the following syntax restrictions:

- A TPF file consists of a series text lines of keywords and subkeywords.
- The first word in the text line must be a keyword or subkeyword, unless the text line contains a comment.
- A line starting with # in front is a comment.
- Each keyword defines the scope for any subsequent text lines until another keyword is encountered.
- Subkeywords are valid within the scope of its preceding keyword.
- Keywords and subkeywords are case-sensitive.

Supported Keywords

PATTERN

FEATURE

PATTERN

Begins a test pattern definition with title *pattern_name*, which will be imprinted on the top of the output GDS file. PATTERN should be specified prior to using any other keywords.

Usage

PATTERN *pattern_name*
dbgrid *db_grid*
designgrid *design_grid*
topcellname *top_cell_name*
gdsfile *gds_file_name*
ssfile *ss_file_name*
markers {none | sem | oddeven | cross}
border { 0 | 1 }
blocksize *val*
inset *val*

Subkeywords

- dbgrid *db_grid*

  Database grid, in user units (microns), in which to specify the test pattern features.

- designgrid *design_grid*

  Design grid on which to snap data, specified in user users (microns). Must be a multiple of dbgrid.

- topcellname *top_cell_name*

  The name of the top cell to create for this test pattern.

- gdsfile *gds_file_name*

  The output GDS file to write, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application.

- ssfile *ss_file_name*

  The output sample spreadsheet file to write, specified as either an absolute path *or* relative to the directory from which you invoke the Calibre WORKbench application.

- markers {none | sem | oddeven | cross}

  Specifies the type of SEM alignment markers to use.

- border { 0 | 1 }

  Sets a border around the pattern.

- blocksize *val*

  Sets the size of each pattern block in microns. (default: 7.0 microns)

- inset *val*

  Sets the inset in microns. (default: 1.5 um)

FEATURE

Begins the definition of an array of features called *feature_type*.

Usage

FEATURE *feature_type* [ROW *number*] [COL *number*] [NONEWLINE]
Widths {*units*}+
Heights {*units*}+
Gaps {*units*}+
Spaces {*units*}+
(etc.)

where:

- ***feature_type***

  A valid feature type listed in "Feature Types and Associated Subkeywords".

- ROW *number*

  Instructs the test pattern generator to place this feature on the specified row.

- COL *number*

  Instructs the test pattern generator to place this feature in the specified column.

- NONEWLINE

  Instructs the test pattern generator to append this feature to the current row.

Subkeywords

The subkeywords vary according to ***feature_type***.

You must specify one *units* value for each feature in the array. Values for *units* must be real numbers. Non-numerical value for units will result in a blank column.

Note

The value for *units* (in any subkeyword) should never exceed:

(blocksize - 2 * inset)

where blocksize and inset are defined through the PATTERN keyword. A value of 4um is used by default.

- Widths {*units*}+

  Widths of the features, expressed in um.

- Heights {*units*}+

  Heights of the features, expressed in um.

- Gaps {*units*}+

  Gaps between the key structures within each feature, expressed in um.
- Spaces {*units*}+

  Spaces between individual structures within each feature, expressed in um.
- Sizes {*units*}+

  The length of one edge of a square structure within each feature, expressed in um.
- Pitches {*units*}+

  The pitches of the pitch structures, expressed in um.
- Midheights {*units*}+

  The heights of the middle pitch structure(s), expressed in um.
- Cwidths {*units*}+

  CD widths, expressed in um.
- Cheights {*units*}+

  CD heights, expressed in um.
- Hhextensions {*units*}+

  Hammerhead extensions, expressed in um.
- Hhlengths {*units*}+

  Hammerhead lengths, expressed in um.

Feature Types and Associated Subkeywords

iso
Widths fc
Heights

inv_iso
Widths
Heights

iso_pad
Widths
Spaces

iso_pad_single
Widths
Spaces

contact
Sizes

inv_contact
Sizes

contact_line
Sizes
Spaces

dense_contact
Sizes
Spaces

corner
Gaps

pitch
Widths
Pitches
Heights
Midheights

inv_pitch
Spaces
Pitches

inv_line_end
Widths
Gaps

hammer_pad
Widths
Gaps
Hhextensions
Hhlengths

line_end_hammer
Hhlengths
Hhextensions
Gaps
Widths

line_end_crowded
Spaces
Widths
Gaps

line_end
Widths
Gaps

dense_line_end
Widths
Spaces
Gaps

orientation_vh45
Widths {units}+
Spaces {units}+

short_gate
Heights
Bwidths
Bheights
Cwidths
Cheights
Widths
Gaps
Gbgaps
Gggaps

double_u
Iheights
Isizes
Osizes
Vspaces
Hspaces
Iwidths

twotone_iso_contact
Sizes {units}+

rect_h_array
Ewidths {units}+
Eheights {units}+
Spaces {units}+
Counts {units}+
Titles {string}+

iso
Ewidths {units}+
Eheights {units}+
Spaces {units}+
Counts {units}+
Titles {string}+

prox_landing_pad
Widths {units}+
Spaces {units}+

twotone_dense_contact
Widths {units}+
Spaces {units}+

prox_multi_pitch
Widths {units}+
Spaces {units}+

random_logic_3
Widths {units}+
Spaces {units}+

random_logic_1
Widths {units}+
Spaces {units}+

random_logic_2
Widths {units}+
Spaces {units}+

slot
Iwidths
Iheights
Usizes
Hspaces
Vspaces

random_logic_4
Widths {units}+
Spaces {units}+

comb
Widths
Spaces
Eheights

crossover
Widths
Spaces
Eheights

neck

Widths

Spaces

Eheights

Example (Test Pattern)

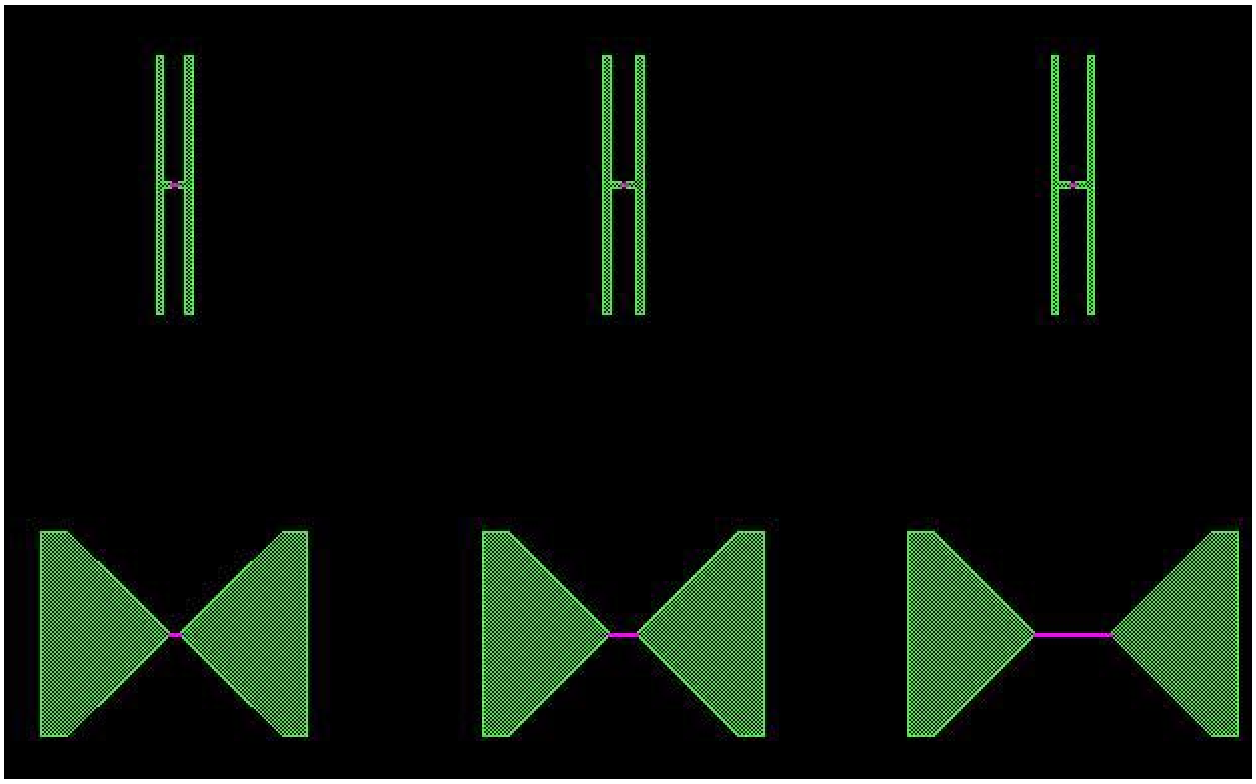
```
# Test Pattern File format, sample 1.
PATTERN "Sample Test Pattern"
  dbgrid .001
  designgrid .001
  topcellname TOP
  gdsfile sample_test_pattern.gds
  ssfile  sample_test_pattern.ss
  markers none
  border  1
FEATURE iso_pad_single
  Spaces     0.17 0.18 0.198 0.248 0.391 0.8
  Widths     0.18 0.18 0.18  0.18  0.18  0.18
FEATURE pitch
  Widths     0.17 0.18 0.198 0.248 0.391 0.8
  Pitches    0.8  0.8  0.8   0.8   0.8   0.8
  Heights    0.36 0.36 0.36  0.36  0.36  0.36
  Midheights 4.0  4.0  4.0   4.0   4.0   4.0
FEATURE inv_pitch
  Spaces     0.17 0.18 0.198 0.248 0.391 0.8
  Pitches    0.8  0.8  0.8   0.8   0.8   0.8
FEATURE line_end_crowded
  Spaces     0.18 0.18 0.18  0.18  0.18  0.18
  Widths     0.18 0.18 0.18  0.18  0.18  0.18
  Gaps       0.17 0.18 0.198 0.248 0.391 0.8
FEATURE hammer_pad
  Widths       0.18 0.18 0.18  0.18  0.18  0.18
  Hhextensions 0.02 0.02 0.02  0.02  0.02  0.02
  Gaps         0.17 0.18 0.198 0.248 0.391 0.8
  Hhlengths    0.06 0.06 0.06  0.06  0.06  0.06
FEATURE line_end_hammer
  Hhlengths    0.06 0.06 0.06  0.06  0.06  0.06
  Hhextensions 0.02 0.02 0.02  0.02  0.02  0.02
  Gaps         0.17 0.18 0.198 0.248 0.391 0.8
  Widths       0.18 0.18 0.18  0.18  0.18  0.18
# End of sample 1.
```

Gauge Object Command Relationships

Gauge Data Objects Defined

Gauge data objects are the internal representation of gauges for a layout. A gauge data object (purple lines in Figure 8-12) measures the critical distance with one or more markers that span the critical distance. Gauge data measurements include the exact width of the markers.

Figure 8-12. Gauges in a Layout



Gauge data objects can also be converted to gauge measurement objects, which contain information on the gauges in relation to the test structures.

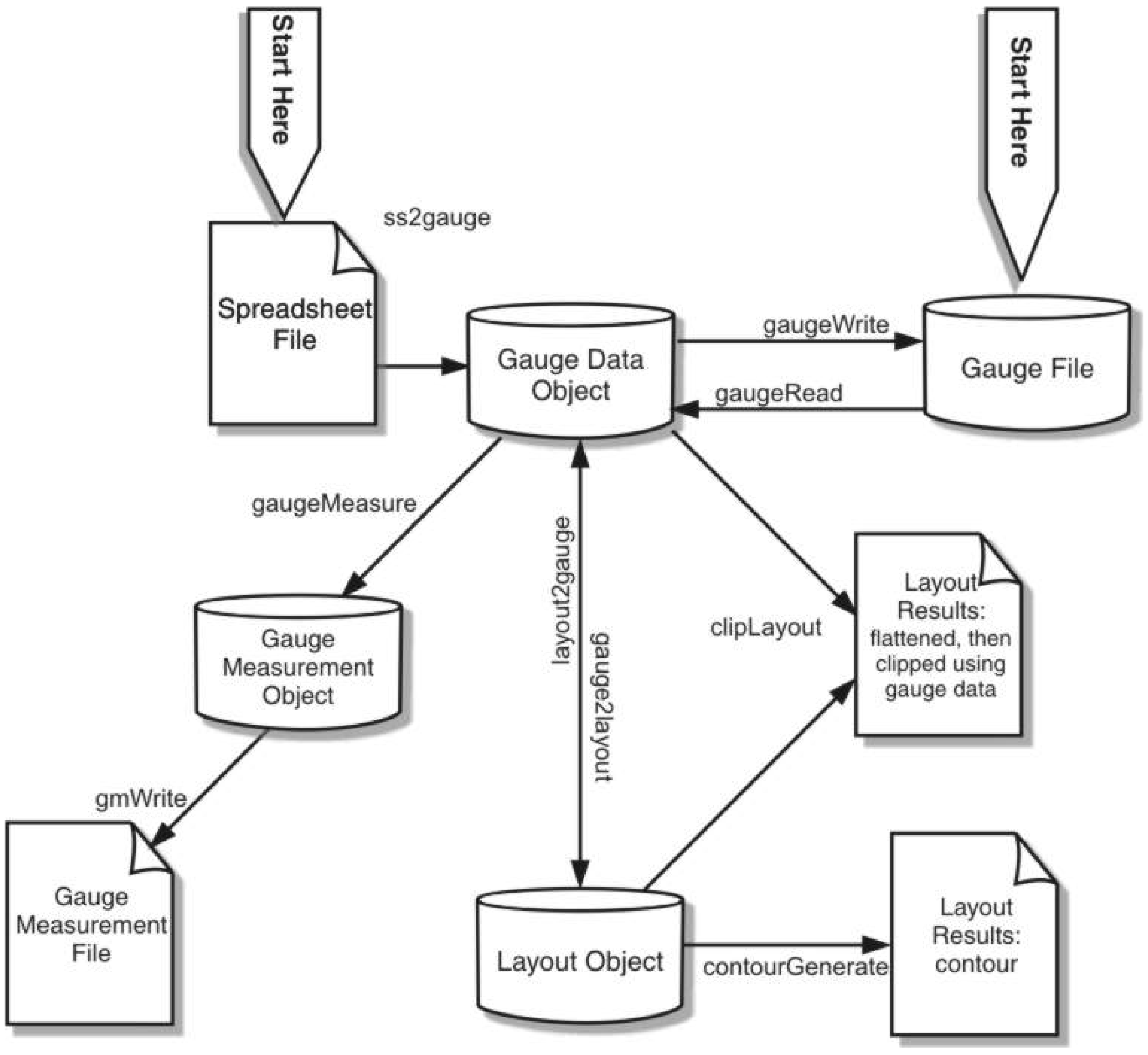
Gauge data objects can also be used in conjunction with dense printimage operations.

Gauge objects differ from spreadsheet simulation sites in the following ways:

- A gauge object is not necessarily centered on the nearest edges.
- A gauge object's location is determined by the coordinates of its start (x1, y1) and end (x2, y2) point.
- Gauge objects can be used for modeling visible etch bias.

The various gauge commands interact as shown in Figure 8-13.

Figure 8-13. Gauge Data Object Relationship



Creating Gauge Data Objects

Use one of the following methods to create gauge data objects:

- Using Spreadsheet Files as a Gauge Data Source (ss2gauge) — You must create a spreadsheet for use with an associated test pattern file.
- Using Gauge Files as a Gauge Data Source (gaugeRead / gaugeWrite) — You can re-use gauge data files between sessions.
- Using a Layout File as a Gauge Data Source (layout2gauge) — You can also manually draw paths to represent gauges.

Using Spreadsheet Files as a Gauge Data Source (ss2gauge)

One of the ways to create a gauge data object is using the ss2gauge command, which takes as inputs a spreadsheet file and the corresponding test pattern.

The source spreadsheet file can be the standard output of the Calibre WORKbench test pattern generator, or one you create using some other tool. It must be in the following format:

```
structure_name row column x y loc drawn other meas weight
```

In the conversion from a spreadsheet object to a gauge object, the gauge-specific arguments to the ss2gauge command (mode, count, and dist) are **added** to the spreadsheet data to form the gauge data object format.

- measured_etch_CD — Support for empirical etch data, used in Visible Etch Bias modeling mode
- simulated_etch_CD — Support for simulated etch data, used in Visible Etch Bias modeling mode
- mode - Sets treatment for sub-gauges, if present: normal (0), average (1), min (2), max (3)
- count - Indicates the number of sub-gauge markers to place at each location left and right of the main gauge (2 * count +1 markers total)
- dist - Specifies the distance between subgauge markers in dbunits

(Note that if you use the Test Pattern Generator in Calibre WORKbench, you will need to fill in the Measurement column with data before you use the ss2gauge command.)

Visible Etch Bias (VEB) is used in conjunction with the CM1 resist model calibration model; both CM1 and VEB are described in the *Dense Modeling Reference*.

Using Gauge Files as a Gauge Data Source (gaugeRead / gaugeWrite)

Gauge data objects can be saved to a gauge data file (using the gaugeWrite command). The gauge data file can be used as input in later sessions (using the gaugeRead command).

A gauge data file uses the following format:

```
flag structure_name row col x1 y1 x2 y2 loc drawn other meas_CD \
simulated_CD meas_etch_CD simulated_etch_CD precision weight mode \
count dist [# comment]
```

The structure_name, row, col(umn), loc(ation), drawn, other, meas(urement)_CD and weight values are standard for a Sample Points Spreadsheet file. The following values are added or modified:

- flag — Is the enable (1)/disable (0) flag.
- x1 y1 x2 y2 — Are the coordinates of the beginning and end of the gauge point in database units.
- simulated_CD — Is the simulated CD/space in database units for resist models.
- meas_etch_CD — Is the measured CD/space for etch model information.
- simulated_etch_CD — Is the simulated CD/space for etch model information.
- precision — Is the accuracy with which measurement data is obtained.
- weight — Is a dimensionless non-negative quantity specifying the weight with which the CD/space data associated with the gauge is used with the cost function during model calibration.
- # — Anything after a hash (#) symbol on a line is considered a comment.

In addition, the mode, count, and dist elements created specifically during gauge data object generation are added to the gauge file.

Using a Layout File as a Gauge Data Source (layout2gauge)

Gauge data objects can be produced from a layout file containing gauges (using the layout2gauge command).

The gauges must be wires (paths) drawn using the Path tool on a separate layer. The wires you draw must obey the following conditions:

- Cross at least two edges in your design (either cross a feature or a gap between two features)
- Be perpendicular to the underlying design polygon
- Be long enough to cross the simulated contours (reference threshold)

Note
The layer containing gauges should not be specified in the setup file.

If you use gauges from a loaded layout, you must ensure that you preserve the paths on the layout when you load it.

If your gauge has subgauges, you must specify the -ssg option in order for Calibre WORKbench to search for them.

Output Options from Gauge Objects

You have the following options once you have a gauge object loaded into Calibre WORKbench:

- Creating a gauge data file (gaugeWrite)
- Producing a Gauge Measurement File (gmWrite)
- Writing Gauges to a Layout Object (gauge2layout)

Producing a Gauge Measurement File (gmWrite)

Gauge measurement files (produced by the gmWrite command) are one of the possible outputs for the gauge object commands. It is an ASCII file that contains information in the following format:

measurement location errorcode

where:

- *measurement* is the measured value
- *location* is 0 for CD, 1 for space measurement
- *errorcode* is the result code for the measurement (a number in the range of 0-6):
    - 0 — no error
    - 1 — no edges intersecting this gauge
    - 2 — too few edges in this gauge
    - 3 — too many intersections intersect this gauge
    - 4 — ambiguous measurement (Calibre WORKbench cannot ascertain whether a space or CD was measured. This can be a case of polygons included inside of each other or a corrupted database.)
    - 5 — nonmatching measurement (CD expected, but space measured or vice versa)
    - 6 — multiple errors (for multiple measurements per gauge)

Writing Gauges to a Layout Object (gauge2layout)

You can produce a visual representation of gauge data on a layout object using the gauge2layout command. Layout objects containing gauge data can be subsequently used to produce two types of specialized layout output: clipped layout files and layouts with contours generated on them.

Clipped Layout Files (clipLayout)

You can use gauge objects as filter points to produce clipped output with the clipLayout command. Any part of the layout not within the vicinity of a gauge object is removed, and the flattened clips are written to a layout object.

Contour Files (contourGenerate)

Similar to clipLayout, contourGenerate uses the gauges in the layout as the clipping selection criteria to narrow down the amount of area to generate aerial image contours for.

Index

Third-Party Information

This section provides information on third-party software that may be included in the Calibre family of products, including any additional license terms.

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/terms_conditions/enduser.cfm

IMPORTANT INFORMATION

USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE. USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software between you, the end user, as an authorized representative of the company acquiring the license, and Mentor Graphics Corporation and Mentor Graphics (Ireland) Limited acting directly or through their subsidiaries (collectively "Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by you and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If you do not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **GRANT OF LICENSE.** The software programs, including any updates, modifications, revisions, copies, documentation and design data ("Software"), are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to you, subject to payment of appropriate license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for your internal business purposes; (c) for the license term; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions.

2. **EMBEDDED SOFTWARE.** If you purchased a license to use embedded software development ("ESD") Software, if applicable, Mentor Graphics grants to you a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESD compilers, including the ESD run-time libraries distributed with ESD C and C++ compiler Software that are linked into a composite program as an integral part of your compiled computer program, provided that you distribute these files only in conjunction with your compiled computer program. Mentor Graphics does NOT grant you any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into your products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

3. **BETA CODE.** Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to you a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form. If Mentor Graphics authorizes you to use the Beta Code, you agree to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. You will contact Mentor Graphics periodically during your use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of your evaluation and testing, you will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements. You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on your feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this section 3 shall survive the termination or expiration of this Agreement.

4. **RESTRICTIONS ON USE.** You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than employees and on-site contractors, excluding Mentor Graphics' competitors, whose job performance requires access and who are under obligations of confidentiality. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it, whether by operation of law or otherwise ("attempted transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable transfer charges. Any attempted transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The terms of this Agreement, including without limitation, the licensing and assignment provisions shall be binding upon your successors in interest and assigns. The provisions of this section 4 shall survive the termination or expiration of this Agreement.

5. **LIMITED WARRANTY.**

   5.1. Mentor Graphics warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LICENSED TO YOU FOR A LIMITED TERM OR LICENSED AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

   5.2. THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

6. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 6 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

7. **LIFE ENDANGERING ACTIVITIES.** NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. THE PROVISIONS OF THIS SECTION 7 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

8. **INDEMNIFICATION.** YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH YOUR USE OF SOFTWARE AS

DESCRIBED IN SECTION 7. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

9. **INFRINGEMENT.**

9.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright or misappropriates a trade secret in the United States, Canada, Japan, or member state of the European Patent Office. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the infringement action. You understand and agree that as conditions to Mentor Graphics' obligations under this section you must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to defend or settle the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

9.2. If an infringement claim is made, Mentor Graphics may, at its option and expense: (a) replace or modify Software so that it becomes noninfringing; (b) procure for you the right to continue using Software; or (c) require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

9.3. Mentor Graphics has no liability to you if infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by you that is deemed willful. In the case of (h) you shall reimburse Mentor Graphics for its attorney fees and other costs related to the action upon a final judgment.

9.4. THIS SECTION IS SUBJECT TO SECTION 6 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

10. **TERM.** This Agreement remains effective until expiration or termination. This Agreement will immediately terminate upon notice if you exceed the scope of license granted or otherwise fail to comply with the provisions of Sections 1, 2, or 4. For any other material breach under this Agreement, Mentor Graphics may terminate this Agreement upon 30 days written notice if you are in material breach and fail to cure such breach within the 30 day notice period. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics' reasonable satisfaction.

11. **EXPORT.** Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export any Software or direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

12. **RESTRICTED RIGHTS NOTICE.** Software was developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable. Contractor/manufacturer is Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

13. **THIRD PARTY BENEFICIARY.** For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth herein.

14. **AUDIT RIGHTS.** You will monitor access to, location and use of Software. With reasonable prior notice and during your normal business hours, Mentor Graphics shall have the right to review your software monitoring system and reasonably relevant records to confirm your compliance with the terms of this Agreement, an addendum to this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet report log files that you shall capture and provide at Mentor Graphics' request. Mentor Graphics shall treat as confidential information all of your information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement or addendum to this Agreement. The provisions of this section 14 shall survive the expiration or termination of this Agreement.

15. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF OREGON, USA, IF YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH OR SOUTH AMERICA. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia (except for Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the Chairman of the Singapore International Arbitration Centre (“SIAC”) to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section 15. This section shall not restrict Mentor Graphics’ right to bring an action against you in the jurisdiction where your place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

16. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

17. **PAYMENT TERMS AND MISCELLANEOUS.** You will pay amounts invoiced, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Some Software may contain code distributed under a third party license agreement that may provide additional rights to you. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

    Rev. 060210, Part No. 227900