



# **Calibre® DESIGNrev Reference Manual**

## **Batch Command Support for Calibre® DESIGNrev™, Calibre® WORKbench™, Calibre® LITHOview™, and Calibre® MDPview™**

Software Version 2012.1

---

**© 2008-2012 Mentor Graphics Corporation**  
**All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

#### **RESTRICTED RIGHTS LEGEND 03/97**

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are 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 provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (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 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.

Telephone: 503.685.7000

Toll-Free Telephone: 800.592.2210

Website: [www.mentor.com](http://www.mentor.com)

SupportNet: [supportnet.mentor.com/](http://supportnet.mentor.com/)

Send Feedback on Documentation: [supportnet.mentor.com/doc\\_feedback\\_form](http://supportnet.mentor.com/doc_feedback_form)

**TRADEMARKS:** The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: [www.mentor.com/trademarks](http://www.mentor.com/trademarks).

# Table of Contents

---

## Chapter 1

<b>Introduction.....</b>	<b>13</b>
Calibre DESIGNrev Overview .....	13
Calibre DESIGNrev Workflow .....	14
Calibre Layout Viewer Key Concepts.....	14
Calibre DESIGNrev Requirements .....	14
Modes of Operation .....	15
Conventions Used in this Manual .....	15

## Chapter 2

<b>Working in the Tcl Environment .....</b>	<b>17</b>
Getting Help on Tcl .....	18
Tcl Versus Tk and What the Distinction Means to You .....	20
Extensions to Tcl .....	21
Things You Must Know About Tcl .....	22
Tcl is Case Sensitive .....	22
Many Characters Have Special Meanings in Tcl .....	22
Tcl Comments Can Be Tricky .....	23
Tcl Objects Have Handles .....	24
Most Database Objects are not Tcl Objects .....	25
Questions and Answers .....	26

## Chapter 3

<b>Getting Started .....</b>	<b>29</b>
Command-Line Invocation Options .....	30
Issuing Tcl Commands in Interactive Mode .....	34
Interactive GUI Mode .....	34
Interactive Shell Mode .....	35
Issuing Tcl Commands in Batch Mode .....	37
Batch Mode .....	37
Batch GUI Mode .....	37
Command Mode .....	37
Tcl and Shared Libraries .....	38
Explorations in Modes of Operation .....	38
Example 1 - Listing the Cells in a Layout in Interactive Shell Mode .....	38
Example 2 - Listing the Cells in an Open Layout in Interactive GUI Mode .....	41
Example 3 - Listing the Cells in a Layout Using Batch Mode .....	44
Explorations In Writing Procedures .....	46
Example 4 - Using a Simple Procedure to Write the Layout Hierarchy .....	46
Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy .....	51
Questions and Answers .....	55

<b>Chapter 4</b>	
<b>Writing Layout Viewer Scripts .....</b>	<b>57</b>
Scripts for Layout Generation and Assembly .....	58
Script 1: Writing a Script to Generate a New Layout. ....	58
Script 2: Setting up a Script to Accept Passed In Arguments. ....	60
Script 3: Debugging and Adding Error Handling .....	63
Script 4: Move Cell Origin .....	68
<b>Chapter 5</b>	
<b>Writing Macros .....</b>	<b>69</b>
How to Write a Macro .....	70
Where To Put Macro Code .....	70
Writing the Command Procedure .....	71
Writing the GUI Plug-in .....	71
Registering the Macro .....	72
Simple Macro Example .....	74
Explorations in Creating Macros .....	75
Example 1: Transforming a Simple Script into a Command Procedure. ....	75
Example 2: Exploring Ways to Obtain User-Supplied Data .....	78
Example 3: Writing a GUI Plug-in Procedure .....	81
Example 4: Displaying a Message to the User .....	83
Putting the Macro on the Menu .....	86
Building In Error Handling .....	87
Setting up Your Macro to Load Automatically .....	90
Complete Code for the Sample Macro .....	91
Macro Examples to Study and Learn .....	93
List All Layers .....	93
DRC EXTERNAL Spacing Check .....	94
Questions and Answers About Macros .....	97
Hierarchical Boolean Operation in DRV .....	98
Loading a File With Layer Colors and Properties .....	100
<b>Chapter 6</b>	
<b>Batch Commands for Layout Manipulation .....</b>	<b>101</b>
Reference Syntax .....	109
cwbWorkBench Object .....	110
\$swb bindKey .....	111
\$swb cget -_layout .....	117
\$swb cget -_layoutView .....	118
\$swb cget -_mainMenus .....	119
\$swb deleteLayoutClibk .....	120
\$swb getMenuPath .....	121
\$swb getRulerFont .....	122
\$swb getSelectionPoint .....	123
\$swb getSelectionRect .....	124
\$swb hasSelectionPoint .....	125
\$swb loadDefaultLayerProperties .....	126
\$swb loadLayerProperties .....	127

## Table of Contents

---

\$cwb reloadLayoutClbk .....	128
\$cwb rulerMeasurements .....	129
\$cwb runMultiRveOnRveOutput .....	131
\$cwb runRveOnRveOutput .....	132
\$cwb selectionCloneToNewLayout .....	133
\$cwb setDepth .....	134
\$cwb setRulerFont .....	135
\$cwb show .....	136
\$cwb showWithLayerFilterClbk .....	137
\$cwb updateDisplay .....	138
\$cwb viewedLayoutClbk .....	139
\$cwb zoomAllClbk .....	140
\$cwb zoomToRectClbk .....	141
Layout Object .....	142
layout all .....	142
layout copy .....	144
layout copy2 .....	147
layout create .....	149
layout delete .....	154
layout droasis .....	155
layout filemerge .....	156
layout merge .....	161
layout overlays .....	165
layout peek .....	166
layoutSetDefaultColors .....	171
layout tbmode .....	172
\$L allowupdates .....	174
\$L ancestors .....	176
\$L AND .....	177
\$L asciout .....	178
\$L bbox .....	179
\$L cellname .....	180
\$L cells .....	181
\$L children .....	182
\$L clips .....	183
\$L clipsout .....	184
\$L connect .....	186
\$L COPY .....	188
\$L COPYCELL GEOM .....	189
\$L create cell .....	190
\$L create clip .....	192
\$L create layer .....	195
\$L create polygon .....	196
\$L create ref .....	198
\$L create text .....	201
\$L create wire .....	203
\$L customLayerDrawOrder .....	205
\$L delete cell .....	206
\$L delete clip .....	207

\$L delete duplicate	208
\$L delete layer	209
\$L delete polygon	210
\$L delete polygons	212
\$L delete ref	213
\$L delete text	215
\$L delete wire	218
\$L disconnect	220
\$L duplicate cell	222
\$L exists cell	223
\$L exists layer	224
\$L expand cell	225
\$L expand ref	227
\$L file	228
\$L flatten cell	229
\$L flatten ref	231
\$L gdsout	233
\$L gridsnap	236
\$L holdupdates	238
\$L import layout	240
\$L instancedbout	242
\$L isLayerEmpty	243
\$L ismodified	244
\$L isOverlay	245
\$L isReferenced	246
\$L iterator (poly   wire   text)	247
\$L iterator (ref   sref   aref)	252
\$L iterator count (poly   wire   text)	256
\$L iterator count (ref   sref   aref)	257
\$L layerconfigure	258
\$L layerFilters	260
\$L layernames	262
\$L layers	264
\$L layoutType	266
\$L libname	267
\$L maxdepth	268
\$L MASK_LAYER_INFO	269
\$L modify origin	270
\$L modify text	271
\$L NOT	273
\$L oasisout	274
\$L options	277
\$L OR	279
\$L pushmarkers	280
\$L query	282
\$L rdbout	288
\$L readonly	289
\$L refcount	290
\$L report	291

## Table of Contents

---

\$L scale .....	292
\$L SIZE .....	293
\$L srefsFromAref .....	295
\$L textout .....	297
\$L topcell .....	298
\$L transcript .....	299
\$L transcript output .....	300
\$L transcript range .....	301
\$L transcript state .....	302
\$L units database .....	303
\$L units microns .....	304
\$L units user .....	306
\$L update .....	308
\$L viacell .....	309
\$L XOR .....	310
Peek Object .....	311
\$P delete .....	311
\$P file .....	313
\$P peek .....	314
Overlay Object .....	318
\$O layouts .....	318
\$O layoutHandle .....	319
\$O overlayCells .....	320
\$O overlayout .....	321
Macros .....	322
Macros create .....	322
Macros delete .....	323
Macros menu .....	324
Macros show .....	325
StringFeature Object .....	326
StringFeature .....	326
\$str addToLayout .....	328
cwbLayoutView Object .....	329
\$V cell .....	329
\$V databaseToScreenUnits .....	330
\$V depth .....	331
\$V exportView .....	332
\$V micronToScreenUnits .....	333
\$V screenToDatabaseUnits .....	334
\$V screenToMicrons .....	335
version .....	336

## Appendix A

<b>Example Script .....</b>	<b>337</b>
Overview .....	337
Opening a Data File for Writing .....	338
Determining the Layers and Topcell .....	339
Writing a Procedure .....	339

Opening a Data File for Viewing . . . . .	341
Review of Executed Commands . . . . .	342
Creating a Batch Script . . . . .	343
Creating a Macro . . . . .	347

**Index****Third-Party Information****End-User License Agreement**



## List of Figures

---

Figure 2-1. Manpages Available on the Internet . . . . .	19
Figure 3-1. Interactive GUI Mode . . . . .	34
Figure 3-2. Graphical Representation of a Recursive Procedure. . . . .	54
Figure 5-1. File Browser Dialog Box Popup . . . . .	79
Figure 5-2. The Updated File Browser Widget . . . . .	80
Figure 5-3. The Sample Message Box. . . . .	83
Figure 5-4. Write Permission Error (write_hier.tcl) . . . . .	88
Figure 6-1. Added Menu . . . . .	121
Figure 6-2. Merge and Preserve. . . . .	158
Figure 6-3. Relating User and Database Units to the Design Grid . . . . .	307

## List of Tables

---

Table 2-1. DESIGNrev Handles .....	27
Table 3-1. Summary of Modes .....	29
Table 3-2. Invocation Options .....	30
Table 6-1. Macro Commands .....	102
Table 6-2. Database Objects .....	102
Table 6-3. cwbWorkBench Database Commands .....	102
Table 6-4. Layout Database Commands .....	103
Table 6-5. Peek Commands .....	107
Table 6-6. Overlay Commands .....	107
Table 6-7. StringFeature Database Commands .....	108
Table 6-8. cwbLayoutView Database Commands .....	108
Table 6-9. Syntax Conventions .....	109
Table 6-10. Supported Keyname Formats .....	111
Table 6-11. Actions You Can Bind To Keys .....	111

## List of Examples

---

Example 4-1. Completed Script (no_error.tcl) .....	67
Example 5-1. Message DisplayCode (write_hier.tcl) .....	85
Example 5-2. Completed Code (write_hier.tcl).....	91
Example 5-3. List All Layers.....	93
Example 5-4. DRC External Spacing Check.....	94
Example 5-5. Hierarchical Boolean Operation .....	98



# Chapter 1

## Introduction

---

This chapter introduces the Calibre DESIGNrev batch commands, and contains the following sections related to using the product:

<b>Calibre DESIGNrev Overview .....</b>	<b>13</b>
<b>Calibre DESIGNrev Workflow .....</b>	<b>14</b>
<b>Calibre Layout Viewer Key Concepts .....</b>	<b>14</b>
<b>Calibre DESIGNrev Requirements .....</b>	<b>14</b>
<b>Modes of Operation .....</b>	<b>15</b>
<b>Conventions Used in this Manual .....</b>	<b>15</b>

## Calibre DESIGNrev Overview

Calibre DESIGNrev is a *layout viewer* specifically designed to handle complex designs. The DESIGNrev tool is embedded within other Calibre layout viewers used to view and manipulate layouts. These tools are Calibre WORKbench, Calibre LITHOview, and Calibre MDPview. Each tool provides special-purpose functionality targeted to address their users' needs, plus all of the functionality of the Calibre DESIGNrev layout viewer.

Regardless of whether you are using Calibre DESIGNrev, WORKbench, LITHOview or Calibre MDPview, the information in this manual applies to you. You can invoke any of these four layout viewer applications to run batch commands.

This manual covers only topics related to the Calibre layout viewer Tcl-based batch commands. It introduces you to some of the Tcl and Tk basics you need to know to use the batch commands for Calibre DESIGNrev, WORKbench, LITHOview, and MDPview. It is not designed to teach you Tcl/Tk. It is intended as a starting point, introducing you to topics you may want to explore more deeply through classes, books, and websites available elsewhere.

---

### Note



The Calibre DESIGNrev GUI commands are discussed in the [Calibre DESIGNrev Layout Viewer User's Manual](#).

---

The following topics are discussed in this manual:

- [“Working in the Tcl Environment”](#) on page 17
- [“Getting Started”](#) on page 29
- [“Writing Layout Viewer Scripts”](#) on page 57
- [“Writing Macros”](#) on page 69
- [“Batch Commands for Layout Manipulation”](#) on page 101

The Calibre DESIGNrev GUI commands are discussed in the [Calibre DESIGNrev Layout Viewer User’s Manual](#).

## Calibre DESIGNrev Workflow

Calibre DESIGNrev and the other Calibre layout viewers are designed to be used in conjunction with Calibre Interactive & Calibre RVE when running checks & reviewing check results from most of the Calibre batch tools. Calibre Interactive is a front-end GUI for running Calibre tools in an interactive verification environment, and Calibre RVE is a graphical debug program that interfaces with most IC layout tools.

For information on using the Calibre layout viewers in your verification flow, see the [Calibre DESIGNrev Layout Viewer User’s Manual](#).

## Calibre Layout Viewer Key Concepts

The Calibre layout *viewers* differ from full-featured layout *editors* in terms of the performance and sophistication of key tasks. Layout viewers provide more sophisticated data loading and reporting. Full-featured layout editors provide more sophisticated layout editing.

Calibre DESIGNrev was designed specifically to streamline tasks related to viewing and inspecting data. It also contains sufficient editing capability for targeted fixes and finalizations on your way to tape-out. Loading design data into Calibre DESIGNrev is much faster than loading the same data into a full-featured layout editor.

Calibre DESIGNrev also provides a powerful scripting capability with Tcl, which gives you full custom automated chip-finishing capabilities and allows you to generate text and spreadsheet-based reports about your designs.

## Calibre DESIGNrev Requirements

To run any of the Calibre layout viewers, you must have the license file required by the tool. Calibre DESIGNrev, WORKbench, LITHOview, and MDPview each require a different license. Refer to the [Calibre Administrator’s Guide](#) for more information on licensing these products.

## Modes of Operation

You can run Calibre DESIGNrev, WORKbench, LITHOview, and MDPview in any of five modes, and you can use batch commands in all of these modes:

- Interactive GUI Mode
- Interactive Shell Mode
- Batch Mode
- Batch GUI Mode
- Command Mode

“[Getting Started](#)” on page 29 discusses these modes and how to issue batch command in each mode.

## 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 the Calibre layout viewer applications. 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 you have access to the same extensions used when creating the instances. Within this document, instance commands are also displayed in **blue**. They are active links and take you to the reference pages containing complete command syntax.

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

<b>Bold</b>	A bold font indicates a required argument
[ ]	Square brackets enclose optional arguments. For Tcl syntax examples, some brackets represent evaluated expressions.
<i>Italic</i>	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 the braces are required.
	A vertical bar indicates an either/or choice between items.
...	Braces including ellipses indicate the enclosed argument(s) may be repeated one or more times.
<sup>2</sup>	A superscript value outside square brackets indicates the enclosed argument can be repeated up to that many times.
*	An asterisk outside square brackets indicates the enclosed argument is optional and can be repeated as many times as needed.



# Chapter 2

## Working in the Tcl Environment

---

<b>Getting Help on Tcl</b> .....	<b>18</b>
<b>Tcl Versus Tk and What the Distinction Means to You</b> .....	<b>20</b>
<b>Extensions to Tcl</b> .....	<b>21</b>
<b>Things You Must Know About Tcl</b> .....	<b>22</b>
Tcl is Case Sensitive .....	22
Many Characters Have Special Meanings in Tcl .....	22
Tcl Comments Can Be Tricky .....	23
Tcl Objects Have Handles .....	24
Most Database Objects are not Tcl Objects .....	25
<b>Questions and Answers</b> .....	<b>26</b>

The batch commands for Calibre DESIGNrev, WORKbench, LITHOview, 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 you take advantage of one or more of the excellent books and websites on the language. The following list 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 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, Ken Jones, Jeffrey Hobbs, 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

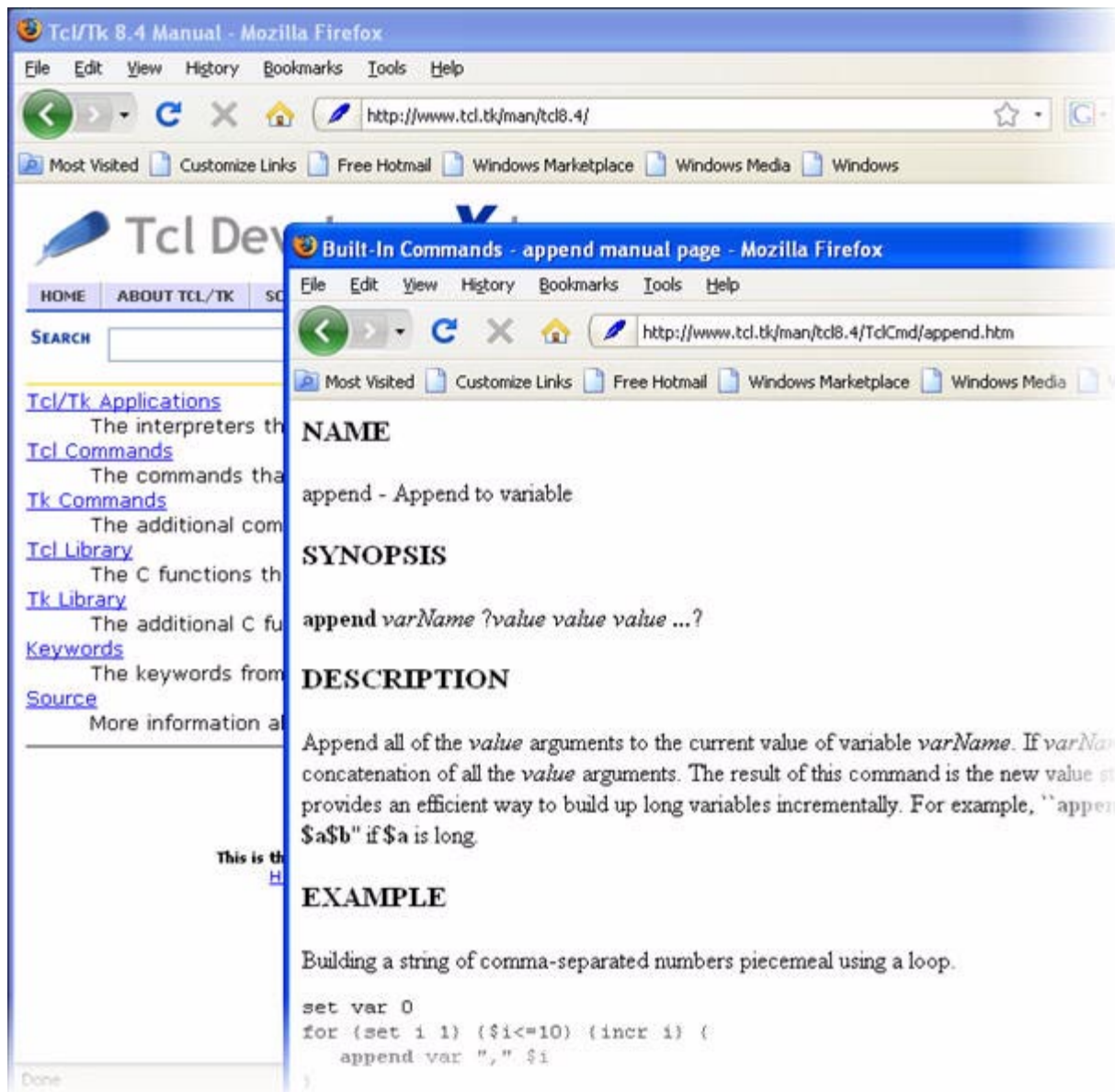
- [Tcl Developer Xchange](http://www.tcl.tk/scripting/) — <http://www.tcl.tk/scripting/>
- [TclTutor](http://www.msen.com/~clif/TclTutor.html) — <http://www.msen.com/~clif/TclTutor.html>
- [Beginning Tcl](http://wiki.tcl.tk/298) — <http://wiki.tcl.tk/298>
- [The Tcler's Wiki](http://wiki.tcl.tk) — <http://wiki.tcl.tk>
- [ActiveState.com](http://www.activestate.com/Products/ActiveTcl/) — <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/>
- Find reference pages for the batch commands in “[Batch Commands for Layout Manipulation](#)” on page 101 of this manual.

Figure 2-1. Manpages Available on the Internet



## 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 DESIGNrev, WORKbench, LITHOview, 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 DESIGNrev, WORKbench, LITHOview, 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.

---

The .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 existing in the user’s home directory and the current directory.

Because many .signaext files contain Tk commands<sup>1</sup> (though you may not be aware of it) and because the tools process the .signaext file every time they invoke, users experience difficulties when they attempt to invoke Calibre DESIGNrev, WORKbench, LITHOview, 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 instructing 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:

```
# protected Tk command
if [isTkLoaded] {
    set filename [tk_getSaveFile]
}
```

---

**Note**

The Unix home directory `.signaext` is sourced first and then the local directory `.signaext` file is sourced. This is important to know, as the local definition may supersede a Unix home definition.

---

## Extensions to Tcl

One of the reasons Tcl/Tk is used so widely is that it 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 DESIGNrev, WORKbench, LITHOview, and MDPview mainly use one:

- **iTcl/iTk**, also called Incr Tcl, is an extension to Tcl providing object-oriented programming capabilities. You can learn more about iTcl/itk at <http://incrtcl.sourceforge.net/itcl/>.

Because Calibre DESIGNrev, WORKbench, LITHOview, and MDPview use the iTcl/iTk extension, it is available for use in any scripts you write to run with these applications.

The layout viewer applications also rely on proprietary extensions created by Mentor Graphics Corporation. The 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 Tk be available and can only be in one of the GUI modes. Refer to the individual reference pages to determine 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.

---

### Note



All Tcl statements must start with a command.

---

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 [“Batch Commands for Layout Manipulation”](#) on page 101, 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 following list 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 containing 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 displaying 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. 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."\n# -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.
- User reference names *versus* DESIGNrev instance assigned names. For example, the command “`layout create mylayout.gds`” creates all of the following items:
  - *layout* is the type of object you are creating in DESIGNrev memory.
  - *layout0* is the system generated name (handle) for the *first* layout loaded into memory.
  - *mylayout.gds* is the name of the *input* layout data file to be loaded into memory.
  - *mylayout* can be the name by which you generally refer to the layout.

---

### Note



The “`layout create mylayout.gds`” command doesn’t actually *create* the layout, it loads it into memory, while the command, “`layout create`”, with no filename does actually create an empty layout in memory.

---

- Names used as part of a command name *versus* those that are not:
  - The **Object Class Commands** create and manipulate layout objects all begin with the string “*layout*”.
  - The **Instance Commands** manipulate the contents of a specific layout all begin with the layout’s handle. For example, “`layout0 create cell mytopcell`”.



## 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 layout handles?**

A: When you create an object, the command returns the layout 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 layout handles?**

A: For objects that do not have layout 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 requiring that I know the layout handle. How can I get it?**

A: The following code will save the layout handle in the variable “layout”.

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

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

A: You can use the built-in handles in the table below, but with caution. Preferably, capture handles as the application returns them, and assign them to a variable, to guarantee you are pointing to the correct objects. Using built-in handles should be avoided in batch mode.

Here are the DESIGNrev built-in handles:

**Table 2-1. DESIGNrev Handles**

Object	Handle
window	::cwbWorkBench::cwbWorkBench <i>n</i>
layouts	layout <i>n</i>
overlays	overlay <i>n</i>
views	::cwbLayoutView <i>n</i>

Where *n* is 0 through the number of active handles, minus one.

---

**Note**

Overlay objects behave like layout*n* objects, however they have a few more operations and are restricted in their use of edit commands.

---



## Chapter 3 Getting Started

---

You can run Calibre DESIGNrev, WORKbench, LITHOview, and MDPview in any of five modes:

- [Interactive GUI Mode](#)
- [Interactive Shell Mode](#)
- [Batch Mode](#)
- [Batch GUI Mode](#)
- [Command Mode](#)

---

**Note**

Used alone, the commands **calibredrv**, **calibrewb**, **calibreiv**, and **calibremdp** invoke the associated applications in interactive GUI mode.

---

**Table 3-1. Summary of Modes**

Mode	Displays GUI?	Uses Tk?	Invocation	Comments
Interactive GUI	yes	yes	(no arguments)	Can see the data.
Interactive shell	no	no	-shell	Can interact with the application.
Batch	no	no	<script>	Can only use Tcl and non-GUI commands.
Batch GUI	no	yes	<script> -gui	Can use batch commands requiring Tk.
Command	no	no	-a <command>	Can only use Tcl and non-GUI commands.

---

**Caution**

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

---

## Command-Line Invocation Options

You can supply optional command switches as described in [Table 3-2](#) while operating in different modes.

**Table 3-2. Invocation Options**

Execution Mode	Option	Description
Interactive Shell	-shell	Specifies to use the shell interface instead of the GUI. To interact with the program you must type commands into the shell.
Interactive Shell and Interactive GUI	-s <i>script</i>	Specifies a Tcl file to load and evaluate at startup time. Evaluation is done after the GUI is fully functional. A cwbWorkBench object exists.
	-noedit	Specifies to run the GUI in view-only mode. Turns off all layout commands that modify the layout.

**Table 3-2. Invocation Options (cont.)**

Execution Mode	Option	Description
Interactive GUI	-dl <i>layerprop_file</i>	Specifies a default <i>layer properties</i> file.
	-m <i>layoutfile</i> *	Specifies an optional startup layout file to be loaded on invocation; accepts GDS or OASIS files. If loading multiple files, each file name must be specified with its own -m option. Cannot be used with the -v option, unless the -v option is used without <i>drlayoutfile</i> .  <b>Note:</b> If a startup layout file is specified without the -m option, the viewer automatically assumes the file is a Tcl macro script and passes it to the Tcl interpreter, which will generate an error.
	-v [ <i>drlayoutfile</i> ]	Specifies an optional startup OASIS layout file to be loaded as a view-only disk-resident file. If a disk-resident index file is not present, one is created. Cannot be used with the -m option, unless the -v option is used without the optional <i>drlayoutfile</i> .  If <i>drlayoutfile</i> is left omitted, all OASIS files will be read as disk resident. It is not possible to read OASIS files as memory-resident if <i>drlayoutfile</i> is omitted. It must be set to read overlay files created with disk resident OASIS files.  <i>Not available in Calibre DESIGNrev (MDPview viewers and above).</i>
	-g <i>goto-cell</i>	Specifies an optional cell to be viewed after layouts have been loaded. Can be used with the -m option. Cannot be used with the -v option.
	-l <i>layerprop_file</i>	Loads a single <i>layer properties</i> file to be used with the layout filename specified by the -m or -v option.
	-b <i>bookmarkname</i>	Loads a previously-created bookmark file.
	-hideLayers	When specified with the -m option, causes all the layers to start as set to hidden to speed up initial load time. Overrides any other related preference settings for this session only.

**Table 3-2. Invocation Options (cont.)**

Execution Mode	Option	Description
Interactive GUI	-endDepth <i>depth</i>	Starts with the end depth set to the specified value. Overrides other related preference settings for this session only.
	-o <i>overlay_file</i>	Loads a previously-created overlay file (.ovly) during invocation.
	-rve [ <i>"rveFile rveArgs"</i> ]	Invokes Calibre RVE and loads an optional rve file specified by rveFile. If no rveFile is specified, RVE prompts for a filename. If <i>rveArgs</i> are also specified, the entire <i>rveFile rveArgs</i> specification must be enclosed in quotes.
Batch	script_name -- script_arg1, script_arg2, ...script_argn	The presence of script_name as the first argument places the tool into script mode. The tool executes the script_name file and then exits. The "--" (double dash) tells the tool to stop parsing arguments to the right of these characters; the script arguments are passed directly to the script's argc and argv variable without being interpreted.
Batch GUI	-gui	Evaluates the script with the Tk environment and the application GUI loaded and available for use by the script.
Command	-a <i>command</i>	Evaluates a single Tcl command and then exits. May require special treatment to retain output; see <a href="#">"Command Mode"</a> on page 37.
All	-threads <i>num_thread</i>	Specifies the number of threads to use in parallel macros. By default, the system determines and uses the maximum number of CPUs automatically. <i>Not available in Calibre DESIGNrev.</i>
	-tb <i>classic   origin   ignore   extent</i>	Specifies how the extent of text affects the calculation of a cell bounding box: <ul style="list-style-type: none"> <li>• classic — Text origin is used as lower-left coordinate of the extent. The height is set fixed to 0.25 microns independent of the actual text attributes or user units of the design. The contribution of the width is zero regardless of number of characters in text. The text-extent is added to the cell-extent.</li> <li>• origin — The text origin is used as lower-left and upper-right coordinate of the extent. The origin of the text is added to the cell-extent.</li> </ul>



**Table 3-2. Invocation Options (cont.)**

Execution Mode	Option	Description
All	-tb <i>classic</i>   <i>origin</i>   <i>ignore</i>   <i>extent</i> (continued)	<ul style="list-style-type: none"> <li>ignore — The text does not contribute to the cell-extent.</li> <li>extent — The text extent in the database is determined mainly by the height of text. The text-width is calculated by using the character length multiplied by a fixed ratio of the height. The ratio is given by the font geometry of the application. Because of the under stroke of certain characters like “g” etc. the lower-left coordinate can be slightly lower than the origin of the text. If the text is rotated or aligned, the extent is calculated taking these attributes in account also. The text-extent is added to the cell-extent.</li> </ul>
	-help -h	Prints command usage.
	-colorByDefinition	Causes all layers within each jobdeck chip definition to be colored the same.
	-colorByPlacement	Causes the application to start in a mode where unique jobdeck placement definitions are assigned different colors.
	-version	Print the version number of the application.
	-nowait	Indicates to <i>not</i> wait if all licenses are unavailable. The layout viewer exits if it cannot get a license.
	-wait <i>time</i>	<p>By default, a Calibre tool waits indefinitely for a license to become available. With the -wait option, a tool waits for any appropriate license for the number of minutes specified by the <i>time</i> argument; the tool then exits if no licenses become available before <i>time</i> expires.</p> <p>Licenses can be acquired from this product or from any of the tool’s parent products. The order of precedence, from lowest to highest, is:</p> <ol style="list-style-type: none"> <li>1. Calibre DESIGNrev</li> <li>2. Calibre MDPview</li> <li>3. Calibre LITHOview</li> <li>4. Calibre WORKbench</li> </ol> <p>License checking occurs in two stages. The first pass checks for a qualified license without a wait. If no license is acquired, a second pass is performed and a wait is done on the first existing qualified license.</p> <p>For example, Calibre MDPview can use a Calibre MDPview, LITHOview, or WORKbench license, if one is available, but it cannot use a DESIGNrev license. Additionally, there is not a MDPview license, Calibre can wait for a LITHOview or WORKbench license if one exists.</p>

## 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:

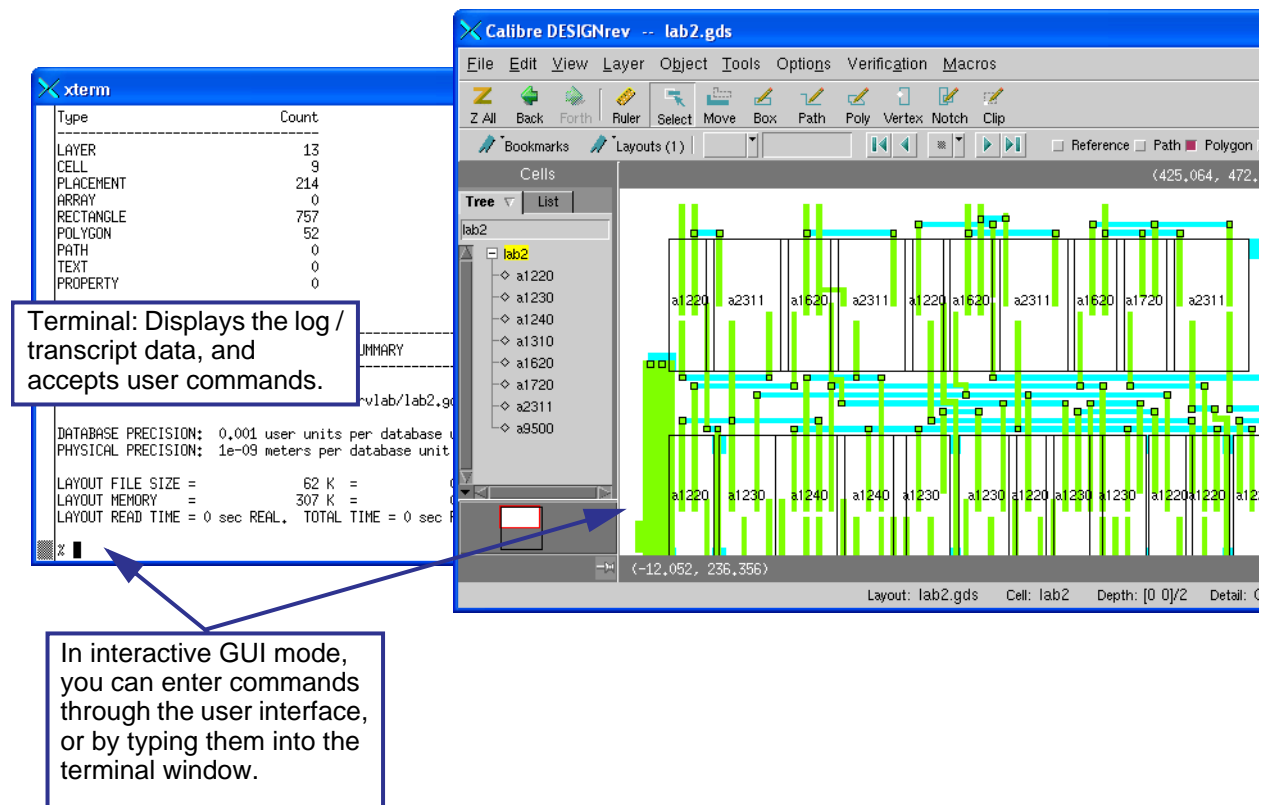
- [Interactive GUI Mode](#) — Viewer and shell window. Default mode.
- [Interactive Shell Mode](#) — Shell window only. Invoked with the `-shell` option.

### 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 layout viewer command without a script name or the `-a` or `-shell` arguments.

Figure 3-1. Interactive GUI Mode



## Running RVE From the Command Line

As specified in [Table 3-2](#), 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. 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 layout viewer command with the *-shell* argument.

```
/labs/drvtlab % calibredrv -shell
```

When you launch the application, the shell window prompt changes to the Tcl “%” prompt:

```
/labs/drvtlab % calibredrv -shell

//  Calibre DESIGNrev v2010.3 Wed Jun 17 06:59:21 PDT 2010
//
//          Copyright Mentor Graphics Corporation 1999-2010
//          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

%
```

The application writes transcript information to the shell window as it would in GUI mode:

```
% set layout [layout create lab2.gds]
Collecting data...
Analyzing data...
Sorting data...

Type                                     Count
-----
LAYER                                   13
CELL                                    9
PLACEMENT                             214
ARRAY                                   0
RECTANGLE                             757
POLYGON                                52
```

```
PATH                                0
TEXT                                0
PROPERTY                            0

GDS FILE READ FROM `/labs/drvlab/lab2.gds'

DATABASE PRECISION:  0.001 user units per database unit
PHYSICAL PRECISION:  1e-09 meters per database unit

LAYOUT FILE SIZE =                62 K =                0 M
LAYOUT MEMORY     =                291 K =                0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.

layout0

%
```

The application also writes the data returned by commands directly to the shell window:

```
% $layout cells
lab2 a9500 a1620 a1220 a2311 a1720 a1230 a1240 a1310
```

## 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.
- **Command Mode** — Tk is not 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 layout viewer commands requiring 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:

- Tk command.
- Calibre layout viewer 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
```

### Command Mode

Command mode is a special mode that evaluates a single Tcl command and then exits. You can use this mode any time you will not instruct the application to execute any Tk commands or any of the Calibre layout viewer commands that require GUI mode.

If a batch command returns a data string, you can explicitly output the returned results to a file or the screen using the **puts** command with a redirect:

```
calibredrv -a puts [layout peek mylayout.gds] >> data.txt
```

## 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 DESIGNrev, WORKbench, LITHOview, and MDPview, specifying a path to the shared library.

## Explorations in Modes of Operation

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

- [Example 1 - Listing the Cells in a Layout in Interactive Shell Mode](#)
- [Example 2 - Listing the Cells in an Open Layout in Interactive GUI Mode](#)
- [Example 3 - Listing the Cells in a Layout Using Batch Mode](#)

### Example 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 the file *sample.gds* exists in the directory from which you invoke the application.

Example steps:

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 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<sup>1</sup> generated when loading the layout, then prints the result of processing<sup>2</sup>, in this case “layout0”.

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

```
% set my_layout [layout create sample.gds]
Collecting data...
Analyzing data...
Sorting data...

Type                                     Count
-----
LAYER                                   24
CELL                                    3
PLACEMENT                              234
ARRAY                                   0
RECTANGLE                              712
POLYGON                                 50
PATH                                    0
TEXT                                    0
PROPERTY                                0

GDS FILE READ FROM '/labs/drvtlab/sample.gds'

DATABASE PRECISION:  0.001 user units per database unit
PHYSICAL PRECISION:  1e-09 meters per database unit

LAYOUT FILE SIZE =          62 K =          0 M
LAYOUT MEMORY    =          291 K =          0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.

layout0

%
```

3. At the “%” prompt, type the following command, then press **Enter**.

1. Whenever you run Calibre DESIGNrev, WORKbench, LITHOview or MDPview, the application generates a transcript containing status information, errors, and warnings.
2. Printing the Tcl results to the terminal window is 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.

```
$my_layout cells
```

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
Overlay — O
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.

Listing cells in shell mode:

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

4. Exit the application.



## Example 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 “[Example 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.

Example steps:

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 viewedLayoutClibk](#) 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 may 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 cwb [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`<sup>1</sup>.

---

#### Note



The examples in these sections assume 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.

---

#### Caution



Even if you are running one of the Calibre layout viewer applications other than Calibre WORKbench, you need to create a *WORKbench object*. **The application window object is called `cwbWorkBench`, whether you are running Calibre DESIGNrev, WORKbench, MDPview, or LITHOview.** Note some of the hardcoded variable names and handles begin with “cwb”. These apply to all the applications, and not just WORKbench.

---

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 it is ready for the next command.

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

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

Here you use the variable `cwb` 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 viewed layout.

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

*Variable substitution* replaces `$cwb` with the value of `cwb`, 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:

---

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>.

```
set my_layout layout0
```

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

```
$my_layout cells
```

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.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set my_layout [$wb cget -_layout]
layout0
% $my_layout cells
TOPCELL a463 a310 b561 c980 981
```

#### Note



In both [Example 1 - Listing the Cells in a Layout in Interactive Shell Mode](#) and [Example 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.

## Example 3 - Listing the Cells in a Layout Using Batch Mode

Topic introduced:

- Displaying results using the **puts** 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 containing only those commands 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 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.

Example steps:

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 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:

```
/labs/wb2 % calibrewb DisplayCellList.tcl
```

```
Collecting data...
Analyzing data...
Sorting data...
```

Type	Count
-----	-----
LAYER	13
CELL	9
PLACEMENT	214
ARRAY	0
RECTANGLE	757
POLYGON	52
PATH	0
TEXT	0
PROPERTY	0

```
GDS FILE READ FROM '/labs/wb2/wblab.gds'
```

DATABASE PRECISION: 0.001 user units per database unit  
PHYSICAL PRECISION: 1e-09 meters per database unit

LAYOUT FILE SIZE = 62 K = 0 M  
LAYOUT MEMORY = 307 K = 0 M  
LAYOUT READ TIME = 1 sec REAL. TOTAL TIME = 1 sec REAL.

top\_level\_cell cell\_g cell\_h cell\_i <— **Results**

/labs/wb2 %

## Explorations In Writing Procedures

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

- [Example 4 - Using a Simple Procedure to Write the Layout Hierarchy](#)
- [Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy](#)

### Example 4 - 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 Calibre DESIGNrev terminal window.

Example steps:

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 Tcl command **open** opens a file and returns the *fileID* 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 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 cwb [find objects -class cwbWorkBench]
set L [$cwb cget -_layout]
```

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

```
puts $fileID "Topcell: [set top [$L 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 **\$L topcell**. This command returns only the name of the topcell, rather than the full list of cells you listed in the previous exercises.

#### Note



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 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 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.

- This formatting follows the recommended standards, and should be used to enhance readability and code clarity.
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 [$L 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 [$L children $top] {  
    foreach child [$L 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 [$L children NAND2A] {  
    list_children $layout $fileID $cell  
}  
foreach cell [$L 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 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 5 - 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 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. Optionally, save the pathname of the invoking tool:

```
info nameofexecutable
/clbr_latest/ic/ixl/Mgc_home/pkgs/icwb/pvt/calibrewb
```

14. Close the file:

```
close $fileID
```

You must always close any file you open to avoid data corruption and system resource depletion.

Here's a transcript of writing the layout hierarchy:

```
/labs/stdtdesign % calibredrv -m std_des.gds

% set fileID [open dump_hier.txt w]
% puts $fileID "This file displays the hierarchy for std_des.gds"
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set layout [$cwb cget -_layout]
layout3
% puts $fileID "Topcell: [set top [$L topcell]]"
% proc list_children { L F C } {
    puts $F "Children of $C: [$L children $C]"
}
% list_children $layout $fileID $top

% foreach cell [$L children $top] {
    list_children $layout $fileID $cell
}
% foreach cell [$L children $top] {
    foreach child [$L 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 5 - 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.

Example steps:

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 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 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:

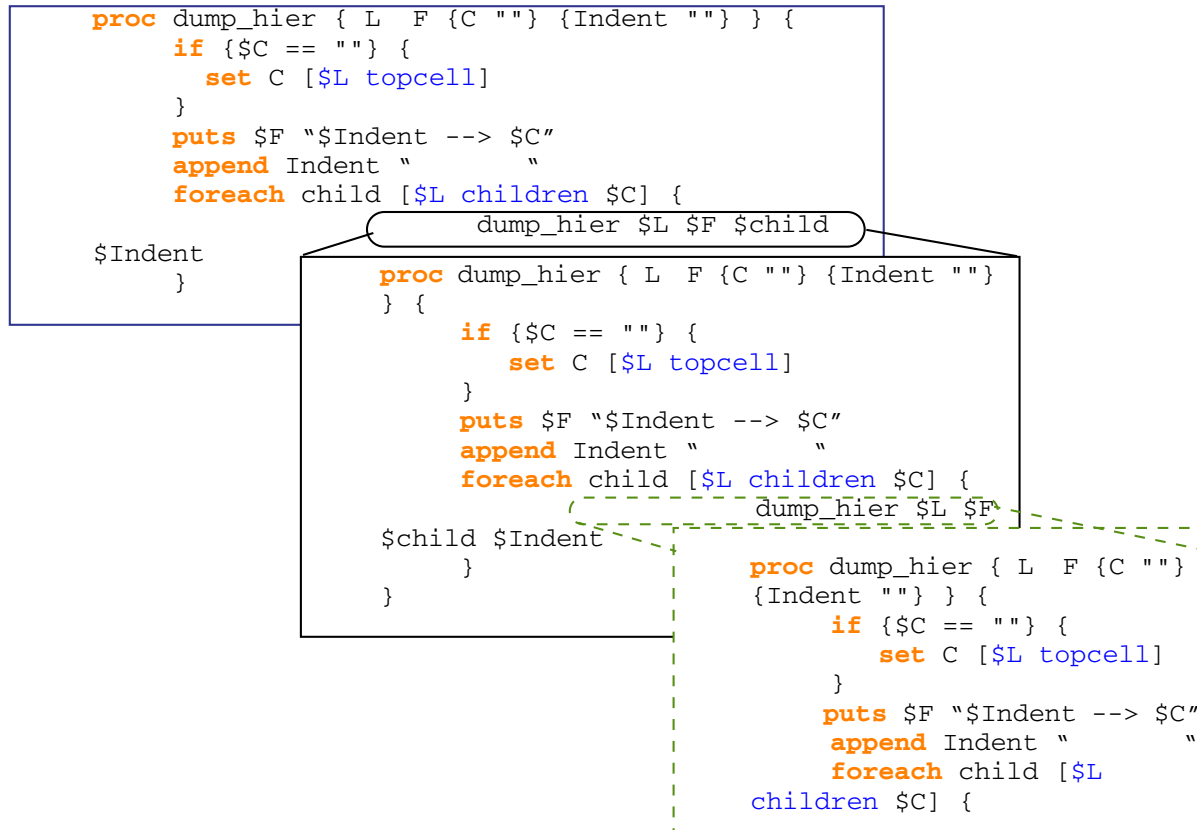
- *Local variables* are variables that exist (have meaning) only within a single procedure. When you execute a procedure, Tcl creates a local variable for each of the arguments to that procedure. While Tcl is processing the body of the `dump_hier` procedure, it knows what `$L` is. Once it exits the procedure,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

---

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

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.

**Figure 3-2. Graphical Representation of a Recursive Procedure**



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 3-2, is a completely different variable from the variable F in the middle procedure, shown outlined with a red solid line in Figure 3-2.

## 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 Calibre WORKbench tool. That means any scripts that run in Calibre DESIGNrev will run in WORKbench. It also means you can invoke either program to work through the exercises.

**Q: This manual says 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 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. Notice 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 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 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 the command would not fit onto one line, and 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.





# Chapter 4

## Writing Layout Viewer Scripts

---

*Tcl scripts* are stand-alone programs you write using the Tcl/Tk language, plus any extensions you have available to you. *WORKbench*<sup>1</sup> *scripts* are Tcl scripts written using the Calibre layout viewer 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 DESIGNrev, WORKbench, LITHOview, 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 DESIGNrev, WORKbench, 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.

Example steps:

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 you want to pass arguments to the script defining 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.

Example steps:

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 passed to a Tcl script.

In Tcl, all variables are strings. The Tcl interpreter recognizes how to handle the values 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.



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).

---

The command “`set my_cell [lindex $argv 0]`” extracts the first argument passed to the script for use as the cell name in the following 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 argument 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:

- o `lindex $argv 1` — return the second argument in the arguments list.
  - o `set layer_num [lindex $argv 1]` — process the command enclosed in square brackets, and store the results in the variable `layer_num`.
  - o `$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 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 (as 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. Recall the index of the last element in a list of length `N` is `(N-1)`. Since the last argument to be passed in is 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 that 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 end of the string 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_file.gds
```

## 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*, and then find & fix them for a more realistic programming experience. You will also anticipate possible user errors and build in error handling for those situations.

Example steps:

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

```
set L [layuot create] <-- Note spelling!
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]
$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 (“layout” misspelled as “layuot”) and stops. It writes an error message to the shell window:

```
% calibredrv error.tcl
Invalid command name "layuot" <-- Type of error
while executing "layout create"
Invoked from within "set L [layout create]"
(file "error.tcl" line 1) <-- Line number
%
```

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
```

Transcript:

```
% calibredrv 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)
%
```

The Tcl interpreter recognizes 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
```

Transcript:

```
% calibredrv error.tcl
Error: layout0 create layer bad layer specification while executing
"$L create layer [set layer1] [lindex $argv 1]]"
(file "error.tcl" line 3)
%
```

The error message this time suggests 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 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.

```
% calibredrv error.tcl
```



```
Wrong number of arguments
<cell name> <layer number> <coords> <file name> ...
```

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:

```
% calibredrv 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)
%
```

This error indicates the Tcl interpreter cannot decipher the `$L create polygon` command. Notice that it does not give you much 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 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.

```
% calibredrv 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)
%
```

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

---

#### Note



Check a Tcl reference book or website for information on conditional statements to help your code anticipate possible error conditions.

---

## Completed Script no\_error.tcl

### Example 4-1. 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: Move Cell Origin

Topics introduced:

- Creating cell references.
- Moving the origin of a cell.
- Setting constants for convenience.

For this example, assume you have a layout, and you want to move the origin of the topcell.

---

### Note



You can also use `$L modify origin` command to change the origin of a specified cell.

---

Example steps:

1. Set a variable for the name of the original layout file, and one for the resulting layout.

```
set mylayout mylayout.gds
set myoutfile myoutfile.gds
```

2. Set variables for the X and Y deltas in user units (most of the time in microns).

```
set mydeltax 12.0
set mydeltay 15.0
```

3. Retrieve the topcell name, and rename it to a temporary name.

```
set L [layout create $mylayout -dt_expand]
set mytopcell [$L topcell]
$L cellname $mytopcell myoldtopcell
```

4. Retrieve the database units (dbu). One micrometer (um) times precision equals one dbu.

```
set mydbunits [$L units user]
```

5. Create a new topcell with the old name.

```
$L3 create cell $mytopcell
```

6. Instantiate the original topcell by the desired shift in database units.

```
$L3 create ref $mytopcell myoldtopcell \
[expr $mydeltax / $mydbunits] \
[expr $mydeltay / $mydbunits] 0 0 1
```

7. Expand the old topcell.

```
$L3 expand cell myoldtopcell
```

8. Write the layout to a file.

```
$L gdsout $myoutfile $mytopcell
```

# Chapter 5

## Writing Macros

---

Macros are application extensions you write to add new functionality to the Calibre DESIGNrev, WORKbench, LITHOview, 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 pulldown menu as a menu item. The application generates the Macros menu and menu items 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:

<b>How to Write a Macro</b> .....	<b>70</b>
<b>Simple Macro Example</b> .....	<b>74</b>
<b>Explorations in Creating Macros</b> .....	<b>75</b>
<b>Macro Examples to Study and Learn</b> .....	<b>93</b>

---

### Note



User-defined macro extensions involve working with the Mentor Graphics extensions to Tcl/Tk. The proprietary programming objects you 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 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 instructing the application to ignore the macro when the Tk software package is not loaded, as is described in [“Tcl Versus Tk and What the Distinction Means to You”](#) on page 20.
- 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 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, 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 needing 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 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.

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 20.

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

- Calling the file containing 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 users.
- Sourcing the file containing the macro code from the shell window, then issuing the **Macros menu** command.

## Simple Macro Example

This example takes a simple Tcl procedure and converts it into a macro available through the layout viewer Macros menu.

We will start with an example Tcl script containing only a **proc**, named *return\_key*, which sets a key binding:

```
proc return_key { } {  
    set cwb [find objects -class cwbWorkBench]  
    $cwb bindKey <Key-F2> cwbHelp Help "Open User Manual"  
}
```

We then add a second **proc**, *return\_key\_plug\_in*, which invokes the command procedure from the GUI plug-in. A GUI plug-in procedure is passed two arguments: the handle for the layout viewer application *and* the handle for the window from which the macro is invoked:

```
proc return_key_plug_in { wbHandle window} {  
    return_key  
}
```

The following Tcl script registers the *original* Tcl procedure as a macro:

```
Macros create "my key bindings" return_key return_key_plug_in
```

Finally, to add the macro to the layout viewer Macros menu, **source** the macro from your \$HOME/.signaext file, and then invoke the layout viewer as always:

```
echo "source macro_file.tcl" >> ~/.signaext  
calibredrv
```

---

### Note



To write a more complicated macro, it is easiest to first write the command script, and then test the script. Following this, rewrite the script into a procedure. Finally, write a GUI plug-in to obtain user input data, and register the macro.

---

## Explorations in Creating Macros

This section explores the process of creating a simple macro that writes the hierarchy of the current layout to a file the user specifies. This macro builds upon the examples in [Working in the Tcl Environment](#), 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 5 - Using a Recursive Procedure to Write a Layout Hierarchy”](#) on page 51.

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.

Example steps:

1. Open `dump_hier.tcl`, from [“Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy”](#) on page 51 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 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.
2. 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
```

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

```
proc write_hier { cwb 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.

4. 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:
#      cwb      = WORKbench object handle
```

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

5. Invoke Calibre DESIGNrev in interactive GUI mode.

```
calibredrv
```

6. Load in the layout of your choice.
7. Switch from the GUI window to the terminal window.
8. 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.

9. Get the handle for the application:

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

10. Invoke the write\_hier procedure, passing it the handle to the application plus the name of a file to create:

```
write_hier $cwb "sample_hier.txt"
```

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

```
more sample_hier.txt
```

For example:

```
% source write_hier.tcl
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% write_hier $cwb_sample_hier.txt
% cat sample_hier.txt
--> tcp_level_cell
--> cell_i
--> cell_g
--> cell_h
```

## Example 2: Exploring Ways to Obtain User-Supplied Data

Topics introduced:

- The Tcl `gets` command.
- The `tk_getSaveFile` command.
- Configuring the file browser widget.

Example steps:

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 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
```

Nothing will occur until you press **Enter**.

6. Press **Enter**.

The shell window tells you what filename you entered:

```
% 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:

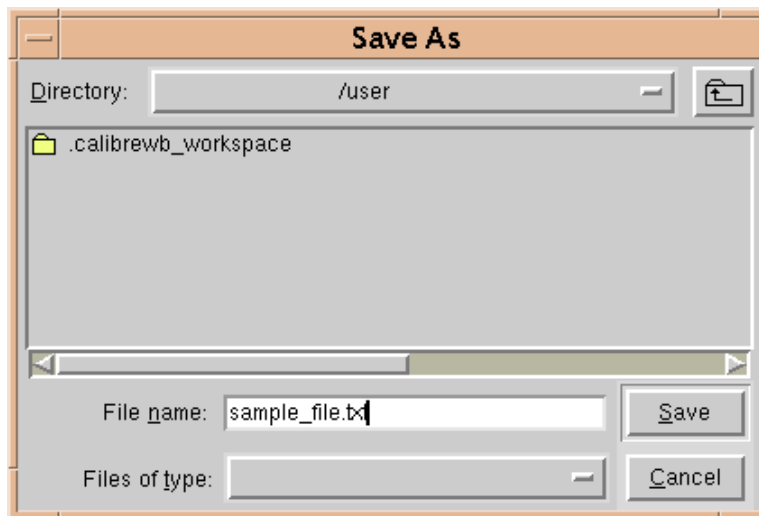
```
set filename [tk_getSaveFile]
puts "You entered $filename"
```

8. Source the script in the terminal window.

```
source browser.tcl
```

Notice that a file browser dialog box pops up.

**Figure 5-1. File Browser Dialog Box Popup**



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
%
```

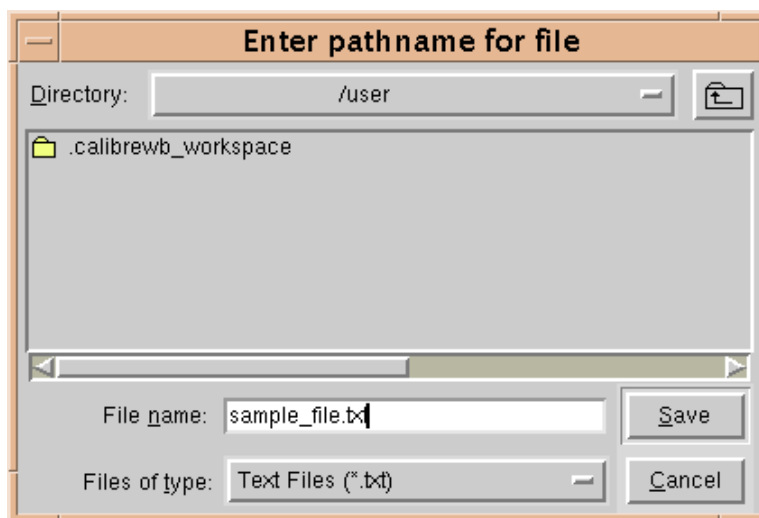
11. Next, modify *browser.tcl* to match the following code.

```
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.

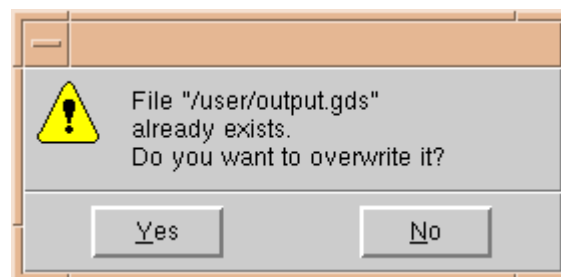
12. Save the file, then source it from the Tcl prompt.

**Figure 5-2. The Updated File Browser Widget**



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 they can simplify writing code to manipulate files.





## 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.

Example steps:

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 layout viewer 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 layout viewer application, and the name of the file it is to create.

4. Add comments to ensure 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 layout viewer 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 read the output data.

## 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 the macro completed successfully.

Example steps:

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 5-3. The Sample Message Box**



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 the command would not fit onto one line, and 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. Message DisplayCode (write\_hier.tcl)

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

proc write_hier { cwb File} {
    set fileID [open $File w]
    set layout [$cwb 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 output
#####

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.

Example steps:

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:

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

The arguments must be supplied in the correct order.

2. Save the file.
3. In the Tcl 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 Enter.

```
Macros menu
```

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

## Building In Error Handling

Topics introduced:

- 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 thus far displays a message box telling you the hierarchy was written successfully to the file. However, what if it wasn't? How would you know, and how would the program know?

Example steps:

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 $File w} errMsg] {  
    tk_messageBox -message "$errMsg"
```

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

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

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

As the Tcl `open` command used with the argument `w` opens the file for writing, the command returns 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*<sup>1</sup> returns an error. If a script executes successfully, `catch` returns 0. If the script does not execute successfully, it returns an error code.

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

---

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

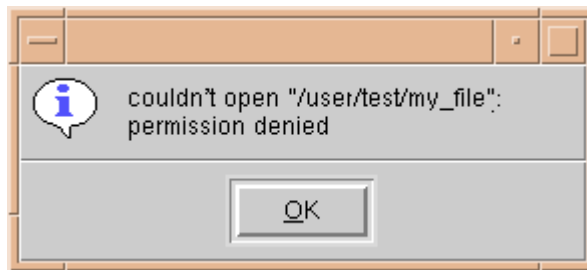
the error message generated by the script. Using this optional argument, you can keep track of the file address, as you did in the original code.

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 $errMsg
```

2. Save *write\_hier.tcl* and test it:
  - a. In the shell window, source *write\_hier.tcl*.
  - b. In the GUI, select “my macro” from the Macros menu.
  - c. When the browser pops up, navigate to a file for which you do not have write permission.
  - d. Select the file and click Save.
  - e. When the macro displays a message box containing the error, click **OK**.

**Figure 5-4. Write Permission Error (write\_hier.tcl)**



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 problem.

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

```
if [catch {open $File w} errMsg] {  
    tk_messageBox -message "$errMsg"  
    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*<sup>1</sup> for the procedure.

---

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.



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
```

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 to 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

Topic introduced:

- The order in which the layout viewer application evaluates scripts.

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

Example steps:

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 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, and 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

### Example 5-2. Completed Code (write\_hier.tcl)

```
#####
# 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 $fileID
        return "file-error"
    } else {
        set layout [$cwb 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

## 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 to the terminal window present in the current layout viewer window. The layout peek command can also be used to list layers, however this macro can be further personalized for specific customer cases.

### Example 5-3. List All Layers

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

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

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

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 in batch Calibre, and returns the results to the open layout session. To activate this macro, insert this code into the *.signaext* file.

### Example 5-4. DRC External Spacing Check

```
#
# INPUT: wbHandle - Handle to a WORKbench object
# window - Window name for WORKbench object
# OUTPUT: Will print out layers in shown layout.
#

proc drc_externalcheck_plug_in {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 "////////////////////////////////////"
```

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

# 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 drc_externalcheck_plug_in
```



## Questions and Answers About Macros

Q: Must I really create two separate procedures to make a macro?

A: No. You can merge the two into a single procedure if you like. You can merge two separate procedures into a single procedure. However, complex macros are easier to build and debug when created as two separate procedures.

Good programming practice is to keep procedures as modular as possible, where each procedure does one thing, and does it well. This enhances maintainability and reduces coding errors.

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
```

## Hierarchical Boolean Operation in DRV

This script performs a boolean operation on a layout within a single cell or within all cells individually.

### Example 5-5. Hierarchical Boolean Operation

```
# booleanLayerOp
#
# Input:
# layout      = DRV layout handle
# cmd         = [OR | AND | NOT | XOR]
# layer1      = Input layer 1
# layer2      = Input layer 2 (Specify "" for single input layer OR)
# outputLayer = Results layer
# cell        = Cell name | "all" (default = "all")
# startDepth  = Start hierarchy depth (default = 0)
# endDepth    = End hierarchy depth (default = "end")
#
proc booleanLayerOp { layout cmd layer1 layer2 outputLayer {cell "all"}
{startDepth 0} {endDepth "end"} } {

    # Validate the existence of the input layers
    # layer2 is optional for some commands
    if { ![ $layout exists layer $layer1] } {
        puts "Error - Non existent input layer1 specified."
        return
    }

    if { $layer2 != "" && ![ $layout exists layer $layer1] } {
        puts "Error - Non existent input layer2 specified."
        return
    }

    # Validate the format of outputLayer
    if { ![isLayer $outputLayer] } {
        puts "Error - Invalid output layer syntax."
        return
    }

    set runerror 0
    set catchStatus ""

    # Perform the operation
    # Note layout copy does not create a layer if it doesn't have
    # any geometries.
    if { $cell == "all" } {
        # Raw layout api boolean operation => No flattening takes place.
        # ie Objects within overlapping instances are treated separately.
        set runerror [catch { $layout $cmd $layer1 $layer2 $outputLayer }
        catchStatus]
    } else {
        if { ![ $layout exists cell $cell] } {
            puts "Error - Invalid cell specification."
            return
        }
    }
}
```

```

    }

    if {$endDepth == "end"} {
        set endDepth 9999
    }
    # Get rid of any predefined "tmp_handle" from previous runs
    # catch {layout delete "tmp_handle"}
    layout copy $layout "tmp_handle" $cell $startDepth $endDepth
    if {[catch {"tmp_handle" $cmd $layer1 $layer2 $outputLayer}
        catchStatus]} {
        set runerror 1
    } else {
        $layout COPYCELL GEOM "tmp_handle" $cell $outputLayer $cell
        $outputLayer
    }
    layout delete "tmp_handle"
}

if {$runerror} {
    puts "Error - ${catchStatus}. Please check:\n1. if input/output
    was correctly entered,\n2. if geometry is available on the input
    layers within current view depth range."
}
}

# Run with...
# set W [find objects -class cwbWorkBench] ;
# Assumes only one cwbWorkBench display exists
# booleanLayerOp layout0 "OR" 1 7 20 "mix" 0 0
# booleanLayerOp layout0 "OR" 1 7 30
# $W updateDisplay

```

## 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]

    # 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

---

This chapter lists all the supported layout manipulation commands available in Calibre DESIGNrev, WORKbench, LITHOview, and MDPview. They are arranged in the following groups:

<b>cwbWorkBench Object</b> .....	<b>110</b>
<b>Layout Object</b> .....	<b>142</b>
<b>Peek Object</b> .....	<b>311</b>
<b>Overlay Object</b> .....	<b>318</b>
<b>Macros</b> .....	<b>322</b>
<b>StringFeature Object</b> .....	<b>326</b>
<b>cwbLayoutView Object</b> .....	<b>329</b>

Within 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
- Peek — P
- Overlay — O
- cwbLayoutView — V
- StringFeature — str

If you write a script that sets the Layout to a variable called \$my\_layout, then to use the [\\$L cells](#) command, you would actually enter the following in your script:

“\$my\_layout cells”

**Table 6-1. Macro Commands**

Object	Brief Description
<a href="#">Macros create</a>	Creates a macro.
<a href="#">Macros delete</a>	Removes (un-registers) the specified macro.
<a href="#">Macros menu</a>	Re-creates the <b>Macros</b> menu with all registered macros.
<a href="#">Macros show</a>	Returns information about the macros currently registered with the application.

**Table 6-2. Database Objects**

Object	Brief Description
<a href="#">cwbWorkBench Object</a>	The application.
<a href="#">Layout Object</a>	A layout.
<a href="#">cwbLayoutView Object</a>	A view of the current layout.
<a href="#">StringFeature Object</a>	A set of polygons in the shape of alphanumeric characters you can place in a layout.
<a href="#">version</a>	Returns the product version string.

**Table 6-3. cwbWorkBench Database Commands**

Command	Brief Description
<a href="#">\$cwb bindKey</a>	Sets a keybinding.
<a href="#">\$cwb cget -_layout</a>	Returns the handle of the layout visible in the layout viewer main window.
<a href="#">\$cwb cget -_layoutView</a>	Queries the cwbWorkBench object for the handle of the layoutView object.
<a href="#">\$cwb cget -_mainMenus</a>	Queries the cwbWorkBench object for the handle of the menus.
<a href="#">\$cwb deleteLayoutClbk</a>	Deletes the currently displayed layout.
<a href="#">\$cwb getMenuPath</a>	Queries the cwbWorkBench object for the handle of the menus.
<a href="#">\$cwb getRulerFont</a>	Gets the ruler font.
<a href="#">\$cwb getSelectionPoint</a>	Gets the selection point.
<a href="#">\$cwb getSelectionRect</a>	Gets the selection rectangle.
<a href="#">\$cwb getSelectionPoint</a>	Gets the selection point.

**Table 6-3. cwbWorkBench Database Commands (cont.)**

<b>Command</b>	<b>Brief Description</b>
\$cwb hasSelectionPoint	Determines if a selection point is set.
\$cwb loadLayerProperties	Loads layer properties from a specified file.
\$cwb reloadLayoutClbk	Reloads the currently displayed layout.
\$cwb rulerMeasurements	Shows information on rulers in the layout.
\$cwb runMultiRveOnRveOutput	Starts Calibre RVE, without closing existing sessions.
\$cwb runRveOnRveOutput	Starts Calibre RVE, a graphical debug program that interfaces with most IC layout tools.
\$cwb selectionCloneToNewLayout	Copies all selected geometry to a new layout object.
\$cwb setDepth	Sets the depth for the Calibre layout viewer object.
\$cwb setRulerFont	Sets the ruler font dynamically.
\$cwb show	Redisplays the layout.
\$cwb showWithLayerFilterClbk	Redraws the display with a filtered set of layers.
\$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.

**Table 6-4. Layout Database Commands**

<b>Command</b>	<b>Brief Description</b>
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.

**Table 6-4. Layout Database Commands (cont.)**

<b>Command</b>	<b>Brief Description</b>
<code>layout droasis</code>	Opens a disk-resident OASIS file in read-only mode.
<code>layout filemerge</code>	Performs a disk-based file merge on multiple GDS input files or multiple OASIS input files without loading them into memory.
<code>layout merge</code>	Merges two layouts, with automatic renaming of the cells to prevent name conflicts.
<code>layout overlays</code>	Returns all defined overlay handles.
<code>layout peek</code>	Returns information about a layout file without loading it.
<code>layout tbmode</code>	Sets the text box extent treatment behavior.
<code>\$L allowupdates</code>	Resumes cell bounding box calculations.
<code>\$L ancestors</code>	Returns unique ancestors for the specified cell.
<code>\$L AND</code>	Performs a BOOLEAN AND on the specified layers.
<code>\$L asciiout</code>	Writes the results in the layout to an ASCII file.
<code>\$L bbox</code>	Returns the bounding box of the specified cell.
<code>\$L cellname</code>	Renames the specified cell.
<code>\$L cells</code>	Returns list of all cells.
<code>\$L children</code>	Returns unique children (but not subsequent descendants) of the specified cell.
<code>\$L clips</code>	Returns clips for a particular cell.
<code>\$L clipsout</code>	Write the currently loaded clips to a file.
<code>\$L connect</code>	Specifies connectivity layers.
<code>\$L COPY</code>	Copies the specified layer.
<code>\$L COPYCELL GEOM</code>	Copies the geometries on a given layer in the specified cell.
<code>\$L create cell</code>	Creates a new cell with the specified name.
<code>\$L create clip</code>	Supports the creation of clips, of which markers are one type.
<code>\$L create layer</code>	Creates a new layer.
<code>\$L create polygon</code>	Creates a polygon with the given coordinates.
<code>\$L create ref</code>	Creates (or places) a reference of the specified cell.
<code>\$L create text</code>	Creates text as specified.
<code>\$L create wire</code>	Creates a path with the given coordinates.
<code>\$L customLayerDrawOrder</code>	Changes the layer drawing order to the specified ordering.
<code>\$L delete cell</code>	Deletes the specified cell and all references to it.



**Table 6-4. Layout Database Commands (cont.)**

<b>Command</b>	<b>Brief Description</b>
\$L delete clip	Deletes the specified clip.
\$L delete duplicate	Deletes the specified duplicate.
\$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 disconnect	Disconnects connectivity layers.
\$L duplicate cell	Duplicates a cell.
\$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 flatten cell	Flattens all levels of hierarchy of the specified cell across all instances of the cell.
\$L flatten ref	Flattens all levels of polygons throughout reference.
\$L gdsout	Writes the layout to a GDS-formatted file.
\$L gridsnap	Depending on the arguments, either snaps all layers and all placements to the specified value of <i>gridsize</i> OR checks for violations of a design grid of <i>gridsize</i> .
\$L holdupdates	Holds cell bounding box calculation.
\$L import layout	Imports the specified layout. Options define the manner in which cell name conflicts are resolved.
\$L instancedbout	Writes out instances of the specified cell into ASCII results database.
\$L isLayerEmpty	Determines if no objects exist on the layer in the layout.
\$L ismodified	Returns 1 if any modifications were made to the layout since the last save.
\$L isOverlay	Returns 1 if the layout is an overlay.

**Table 6-4. Layout Database Commands (cont.)**

<b>Command</b>	<b>Brief Description</b>
<code>\$L isReferenced</code>	Returns 1 if the layout is referenced in one or more overlay objects.
<code>\$L iterator (poly   wire   text)</code>	Returns a list of specified type of geometries.
<code>\$L iterator (ref   sref   aref)</code>	Returns a list of the specified type of references.
<code>\$L iterator count (poly   wire   text)</code>	Returns the number of the specified type of geometries.
<code>\$L iterator count (ref   sref   aref)</code>	Returns the number of the specified type of references.
<code>\$L layerconfigure</code>	Configures a layer with the specified layer properties.
<code>\$L layerFilters</code>	Allows the definition of layer filters used to filter the layers shown in the Layers Browser.
<code>\$L layernames</code>	Returns a list of layer numbers and layer names in the layout.
<code>\$L layers</code>	Returns the list of layers in the layout.
<code>\$L layoutType</code>	Returns or sets the layout file type.
<code>\$L libname</code>	Sets/gets the GDS LIBNAME variable.
<code>\$L maxdepth</code>	Returns the maximum depth of the selected cell.
<code>\$L MASK_LAYER_INFO</code>	Returns the area of shapes on a layer.
<code>\$L modify origin</code>	Changes origin of the specified cell.
<code>\$L modify text</code>	Deletes an existing text object and creates a new one to replace it.
<code>\$L NOT</code>	Performs a BOOLEAN NOT on the specified layers.
<code>\$L oasisout</code>	Writes the layout to an OASIS-formatted file.
<code>\$L options</code>	Checks to see which options were set when the layout was created.
<code>\$L OR</code>	Performs a BOOLEAN OR on the specified layers.
<code>\$L pushmarkers</code>	Moves RVE ASCII database markers to levels relevant to marked polygons.
<code>\$L query</code>	Returns the nearest objects of the specified type.
<code>\$L rdbout</code>	Convert all polygons on a layer into RDB output.
<code>\$L readonly</code>	Controls whether or not the layout can be saved.
<code>\$L refcount</code>	Retrieve number of references to a specified cell or a count of the instances beneath a cell.

**Table 6-4. Layout Database Commands (cont.)**

Command	Brief Description
<a href="#">\$L report</a>	Generate report information about a layout in ASCII format.
<a href="#">\$L scale</a>	Scales the layout by the given value.
<a href="#">\$L SIZE</a>	Sizes the specified layer.
<a href="#">\$L srefsFromAref</a>	Returns a list of SREFs converted from an AREF string.
<a href="#">\$L textout</a>	Returns the text in the specified cell.
<a href="#">\$L topcell</a>	Returns the topcell name.
<a href="#">\$L transcript</a>	Turns recording of transcript information on or off.
<a href="#">\$L transcript output</a>	Writes the transcript to an output file.
<a href="#">\$L transcript range</a>	Returns a range of transcript lines.
<a href="#">\$L transcript state</a>	Queries the transcript state (whether recording is on or off).
<a href="#">\$L units database</a>	Sets the size of database units in meters.
<a href="#">\$L units microns</a>	Sets the size of database units in microns.
<a href="#">\$L units user</a>	Sets the size of user units, expressed as a ratio of user units per database units.
<a href="#">\$L update</a>	Updates the layout by recalculating the bounding boxes, topological ordering, and layer tables.
<a href="#">\$L viacell</a>	Specifies the named cell is a via cell.
<a href="#">\$L XOR</a>	Performs a BOOLEAN XOR on the specified layers.

**Table 6-5. Peek Commands**

Command	Brief Description
<a href="#">\$P delete</a>	Deletes the peek handle defined in a previous layout peek -handle command.
<a href="#">\$P file</a>	Returns the file name for a peek handle.
<a href="#">\$P peek</a>	Displays layout file information for a peek handle.

**Table 6-6. Overlay Commands**

Command	Brief Description
<a href="#">\$O layouts</a>	Returns all layouts within the overlay.
<a href="#">\$O layoutHandle</a>	Returns the handle for the specified overlay layout Id.

**Table 6-6. Overlay Commands (cont.)**

Command	Brief Description
<a href="#">\$O overlayCells</a>	Returns all overlaid cells in the overlay.
<a href="#">\$O overlayout</a>	Saves the overlay as an ASCII setup file.

**Table 6-7. StringFeature Database Commands**

Command	Brief Description
<a href="#">StringFeature Object</a>	Creates a stringfeature.
<a href="#">\$str addToLayout</a>	Inserts a stringfeature into an existing layout.

**Table 6-8. cwbLayoutView Database Commands**

Command	Brief Description
<a href="#">cwbLayoutView Object</a>	Returns the handle for the current view of the layout.
<a href="#">\$V databaseToScreenUnits</a>	Translates a pair of X and Y coordinates from database units to screen units.
<a href="#">\$V depth</a>	Returns the current view depth.
<a href="#">\$V exportView</a>	Outputs a screen capture of the layout viewer.
<a href="#">\$V micronToScreenUnits</a>	Translates a pair of X and Y coordinates from microns to screen units.
<a href="#">\$V screenToDatabaseUnits</a>	Translates a pair of X and Y coordinates from screen units to database units.
<a href="#">\$V screenToMicrons</a>	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 DESIGNrev. 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 and Reference Manual](#) for a description of the available functions.

## Reference Syntax

Table 6-9 identifies the syntax conventions used in this chapter.

**Table 6-9. Syntax Conventions**

Convention	Description
<b>Bold</b>	A bold font indicates a required item.
<i>Italic</i>	An italic font indicates a user-supplied argument.
Underline	An underlined item indicates either the default argument or the default value of an argument.
UPPercase	Uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Square brackets enclose optional arguments. Do not include the brackets when entering the command.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command.
' '	Single quotes enclose metacharacters intended to be entered literally.
	The vertical bar indicates an either/or choice between items. Do not include the bar when entering the command.
...	An ellipsis follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.
<b>Example:</b> \$L iterator { poly   wire   text } cell layer range startRange endRange [-depth startDepth endDepth [-filterCell cellname]]	

## cwbWorkBench Object

Within the Tcl/Tk environment, the `cwbWorkBench` object manages the Calibre layout viewer application. **cwbWorkBench object methods**<sup>1</sup> 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.

---

### Note



Even if you are running one of the layout viewer applications other than Calibre WORKbench, you create a `cwbWorkBench` object to have access to information contained in the application. The application window object is called `cwbWorkBench`, whether you are running Calibre DESIGNrev, WORKbench, MDPview, or LITHOview.

---

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).

---

### Note



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

## \$cwb bindKey

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

DESIGNrev defines a set of standard key bindings and supports custom key bindings using the file `.calibrewb_workspace/keyprefs`. This file contains `bindKey` commands binding keys to predefined DESIGNrev actions or user-written Tcl procedures. These actions allow updating of existing menu accelerators when they exist.

An example template key bindings file, holding some alternate key bindings, is located at `$CALIBRE_HOME/pkgs/icwb/pvt/templates/keyprefs.template`. To make this file more accessible, it is linked into each user's `.calibrewb_workspace` directory when DESIGNrev starts. If you rename it from "keyprefs.template" to "keyprefs", it will load automatically with the layout viewer.

The supported key names are a predefined set of Tcl key names. These are the only keyname syntax formats accepted. Other Tcl variants for the same key names are not recognized. The allowed key names are defined in [Table 6-10](#).

**Table 6-10. Supported Keyname Formats**

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

#### Note



Keys <Control-F1> through <Control-F12> as well as <Shift-F1> through <Shift-F12> are reserved for customer use, and therefore DESIGNrev will not set them. Any other keynames may be set by DESIGNrev.

Calibre currently supports custom key binding of the following actions:

**Table 6-11. Actions You Can Bind To Keys**

Action	Action Performed
<code>cwbCancel</code>	Cancels the current dialog box.
<code>cwbClearContext</code>	Unsets the selected cell reference as the editing context and returns the editing context to the top cell of the layout.

**Table 6-11. Actions You Can Bind To Keys (cont.)**

Action	Action Performed
cwbClearRulerAll	Clears all rulers drawn.
cwbClearRulerLast	Clears the last ruler drawn.
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.
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.
cwbHideAllLayers	Hides all layers.
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.



**Table 6-11. Actions You Can Bind To Keys (cont.)**

Action	Action Performed
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.
cwbOpenLayout	Opens the Choose Layout File 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.
cwbPanDown_small	Micro-pans the layout view down by five percent (can be changed by setting the preference prefs_view_smallPanPercent).
cwbPanLeft	Pans the layout view left by the percentage specified in the preferences.
cwbPanLeft_small	Micro-pans the layout view left by five percent (can be changed by setting the preference prefs_view_smallPanPercent).
cwbPanRight	Pans the layout view right by the percentage specified in the preferences.
cwbPanRight_small	Micro-pans the layout view right by five percent (can be changed by setting the preference prefs_view_smallPanPercent).
cwbPanUp	Pans the layout view up by the percentage specified in the preferences.
cwbPanUp_small	Micro-pans the layout view up by five percent (can be changed by setting the preference prefs_view_smallPanPercent).
cwbPaste	Pastes the contents of the copy buffer to the current location.

**Table 6-11. Actions You Can Bind To Keys (cont.)**

Action	Action Performed
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.
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.
cwbSetSelectionMode	Selects selection mode.
cwbShowAllLayers	Shows all layers.
cwbToggleCellOutline	Toggles the cell outline visibility.
cwbToggleLayouts	Toggles between two loaded layouts. When more than two layouts exist, the normal layout bookmark popup list is displayed.
cwbToggleUserGrid	Toggles the current grid visibility.
cwbUndo	Undoes the most recently executed action.

**Table 6-11. Actions You Can Bind To Keys (cont.)**

Action	Action Performed
cwbUnselectAll	Closes active layout file.
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 by 50 percent.
cwbZoomOut	Zooms out by 50 percent.

## Usage

**\$cwb bindKey** <keyname> {cwbAction | user\_proc | null} [menu\_name "menu\_pick"]

## Arguments

- <keyname>

Is the key combination (as shown in [Table 6-10](#)) to customize. Any key combinations not customized remain as their default values.

### Note



The angle brackets (< >) around the keyname are required.

- *cwbAction*

Is the name of an action to execute (the supported list is shown in [Table 6-11](#)).

- *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) containing 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



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. The quotes (“ ”) are required, and *menu\_pick* must exactly match how the item appears on the menu.

## Returns

A summary of your new key binding.

## Examples

This example binds the F1 key to open the user manual, and adds the “F1” label to **Help > Open User Manual**.

```
set cwb [find objects -class cwbWorkBench]
$cwb bindKey <Key-F1> cwbHelp Help "Open User Manual"
```

This next example erases the F1 keybinding, including removing the “F1” label from the Help menu. It leaves the “Open User Manual” label in place.

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

## Related Commands

None.

## \$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 Calibre layout viewer main window. This command displays only one handle, even if you have loaded multiple layouts into the layout viewer.

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 you are seeking the handle for the layout object, which is a database object.

### Returns

The handle of the current layout, such as “layout0”.

### Example

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

### Related Commands

None.

## \$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 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, such as “::cwbLayoutView0”. If you created different layoutViews for a single layout viewer session, this command returns the handle of the view that has focus.

### Example

```
set wb [find objects -class cwbWorkBench]
$cwb cget -_layoutView
```

### Related Commands

None.

## \$cwb cget -\_mainMenus

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Queries the cwbWorkBench object for the handle of the menus.

---

 **Caution** This is a deprecated command that is no longer necessary. This command should be replaced with the [\\$cwb getMenuPath](#) command.

---

### Usage

**\$cwb cget -\_mainMenus**

### Arguments

- **-\_mainMenus**  
Required keyword indicating you are seeking the handle for the menu object.

### Returns

This command returns the handle of the menu bar path within the cwbWorkBench window.

### Example

None.

### Related Commands

[\\$cwb getMenuPath](#)

## \$cwb deleteLayoutClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Deletes the currently displayed layout. Use this command in scripts working with a displayed layout in the GUI.



#### Caution

The [layout delete](#) command should *only* be used for deleting layouts when running scripts in batch mode without a GUI. If the script is working with a displayed layout in the GUI, use the \$cwb deleteLayoutClbk command or errors will be generated.

---

### Usage

**\$cwb deleteLayoutClbk**

### Arguments

None.

### Returns

None.

### Example

This example deletes the displayed layout, after first changing the displayed layout to the layout corresponding to the specified layout handle.

```
set wb [find objects -class cwbWorkBench]  
$wb viewedLayoutClbk $L  
$wb deleteLayoutClbk
```

### Related Commands

[\\$cwb reloadLayoutClbk](#)

[\\$cwb viewedLayoutClbk](#)

[layout delete](#)



## \$cwb getMenuPath

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Queries the cwbWorkBench object for the handle of the menus.

### Usage

**\$cwb getMenuPath**

### Arguments

None.

### Returns

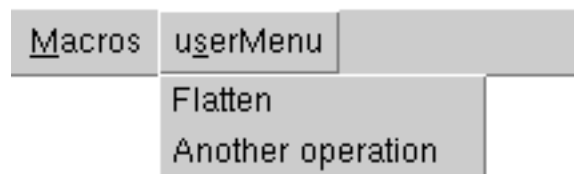
This command returns the handle of the menu bar path within the cwbWorkBench window.

### Example

Here is an example of adding a new menu:

```
# File: userMenu.tcl
#
# Add menu titled "userMenu" to the cwbWorkBench0 menu bar
#
# Initial installation can be with: calibrewb -base -s ./userMenu.tcl
# Needs to be run in each new cwbWorkBench window.
#
set wb [find objects -class cwbWorkBench]
set mm [$wb getMenuPath]
set userMenuButton [menubutton $mm.userMenu -text "userMenu" \
    -menu $mm.userMenu.menu -underline 1]
pack $userMenuButton -side left
set userMenu [menu $userMenuButton.menu -tearoff 0]
$userMenu add command -label "Flatten" -command "puts FLATTEN"
$userMenu add command -label "Another operation" -command "puts ANOTHER"
```

**Figure 6-1. Added Menu**



### Related Commands

None.

## \$cwb getRulerFont

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Returns the ruler font. This is initialized to the Preferences Ruler Font dialog value which is initialized to the preference file value.

### Usage

**\$cwb getRulerFont**

### Arguments

None.

### Returns

Returns the ruler font.

### Example

```
set wb [find objects -class cwbWorkBench]  
$wb getRulerFont  
fixed
```

### Related Commands

[\\$cwb setRulerFont](#)

## \$cwb getSelectionPoint

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Returns a list containing the coordinate of the last selection point used.

### Usage

**\$cwb getSelectionPoint** [**dbu** | **microns**]

### Arguments

- **dbu** | **microns**

An optional keyword indicating the type of units to use for the return data. The default is **microns**. One micron (um) times the precision equals one dbu.

### Returns

Returns the selection point.

### Example

```
% set cwb [find objects -class cwbWorkBench]
% $cwb hasSelectionPoint
1
% $cwb getSelectionPoint dbu
3500 360000
```

### Related Commands

[\\$cwb hasSelectionPoint](#)

## \$cwb getSelectionRect

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Returns a string consisting of either the selected analysis rectangle drawn in the View Center window, or in a window defined by the actual viewed window, if no analysis rectangle has been drawn.

### Usage

**\$cwb getSelectionRect [dbunits | microns]**

### Arguments

- **dbunits | microns**

An optional keyword indicating the type of units to use for the return data. The default is microns. One micron (um) times the precision equals one dbu.

### Returns

Returns a string consisting of {x1 y1 x2 y2}. The default unit for the return data is in microns.

### Example

```
set wb [find objects -class cwbWorkBench]
$cwb getSelectionRect dbunits
```

### Related Commands

None.

## \$cwb hasSelectionPoint

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Determines if a selection point is set.

### Usage

**\$cwb hasSelectionPoint**

### Arguments

None.

### Returns

Returns 1 if selection point is set, or 0 if not set.

### Example

```
% set cwb [find objects -class cwbWorkBench]
% $cwb hasSelectionPoint
0
// Set selection point in GUI
% $cwb hasSelectionPoint
1
% $cwb getSelectionPoint dbu
32000 45000
```

### Related Commands

[\\$cwb getSelectionPoint](#)

## \$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.

### Returns

None.

### Example

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

### Related Commands

[\\$cwb loadLayerProperties](#)

## \$cwb loadLayerProperties

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Loads or reloads the specified layer properties file.

### Usage

**\$cwb loadLayerProperties** *filename*

### Arguments

- *filename*

An optional argument specifying the filename of a layer properties file to load.

### Returns

None.

### Example

```
set wb [find objects -class cwbWorkBench]  
$wb loadLayerProperties test.layerprops
```

### Related Commands

[\\$cwb loadDefaultLayerProperties](#)

## \$cwb reloadLayoutClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Reloads the currently displayed layout. Use this command in scripts working with a displayed layout in the GUI.

### Usage

**\$cwb reloadLayoutClbk**

### Arguments

None.

### Returns

None.

### Example

This example reloads the displayed layout.

```
set wb [find objects -class cwbWorkBench]  
$wb viewedLayoutClbk $L  
$wb reloadLayoutClbk
```

### Related Commands

[\\$cwb deleteLayoutClbk](#)



## \$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

**\$cwb rulerMeasurements**

### Arguments

None

### Returns

Returns ruler information for rulers in the specified layout.

### Examples

```
% set wb ::cwbWorkBench::cwbWorkBench0
% $wb 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 [$wb 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 [$wb 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)
```

### Related Commands

None.

## \$cwb runMultiRveOnRveOutput

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Starts Calibre RVE, and informs you the layout viewer is loading the Calibre RVE graphical debug program. It opens a new RVE session *without* closing existing RVE sessions. This command is identical to the [\\$cwb runRveOnRveOutput](#) in all other respects.

### Usage

**\$cwb runMultiRveOnRveOutput** [*databasefile*]

### Arguments

- *databasefile*  
An optional argument specifying a database file to load. If this argument is not specified, Calibre RVE prompts you for a filename when it loads.

### Returns

None

### Example

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

### Related Commands

[\\$cwb runRveOnRveOutput](#)

## \$cwb runRveOnRveOutput

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Starts Calibre RVE, and transcripts the fact that the layout viewer is loading Calibre RVE.

Executing this command replaces the current opened RVE window with a new RVE window. If you want existing RVE sessions to remain, use the [\\$cwb runMultiRveOnRveOutput](#) command.

---

#### Note



You can only open one RVE window with this command.

---

### Usage

**\$cwb runRveOnRveOutput** [*databasefile*]

### Arguments

- *databasefile*  
An optional argument specifying a database file to load. If this argument is not specified, Calibre RVE prompts you for a filename when it loads.

### Returns

None

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% $wb runRveOnRveOutput
Loading Calibre RVE...
Running RVE...
...
```

### Related Commands

[\\$cwb runMultiRveOnRveOutput](#)

[\\$cwb show](#)

## \$cwb selectionCloneToNewLayout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

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

### Usage

**\$cwb selectionCloneToNewLayout**

### Arguments

None.

### Returns

The new layout handle, such as “layout3”.

### Example

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

### Related Commands

None.

## \$cwb setDepth

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Sets the depth for the Calibre layout viewer object. You will need to follow this command with a [\\$cwb updateDisplay](#) command to see the effects of the command in GUI mode.

### Usage

**\$cwb setDepth** *startDepth endDepth*

### Arguments

- *startDepth endDepth*

Is the starting and ending depth to use for the specified Calibre layout viewer object.

### Returns

None.

### Example

```
set wb [find objects -class cwbWorkBench]  
$wb setDepth 0 3  
$wb updateDisplay
```

### Related Commands

[\\$cwb updateDisplay](#)

## \$cwb setRulerFont

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Sets the ruler font dynamically, updating the Preferences Ruler Font dialog value and the preference value. The preference value is not immediately written to the preference file. This is done at the next normal preference file update, such as exiting from the layout viewer.

### Usage

**\$cwb setRulerFont** *font*

### Arguments

- *font*  
The ruler font.

### Returns

None.

### Example

```
set wb [find objects -class cwbWorkBench]  
$wb setRulerFont fixed
```

### Related Commands

[\\$cwb getRulerFont](#)

## \$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**

### Arguments

None.

### Returns

Redisplays the layout.

### Example

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

### Related Commands

[\\$cwb runRveOnRveOutput](#)



## \$cwb showWithLayerFilterClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Redraws the display with a filtered set of layers. Executing command without a filter name redraws all layers.

### Usage

**\$cwb showWithLayerFilterClbk** *layout\_filter*

### Arguments

- *layout\_filter*  
An optional argument specifying the layout filter designator.

### Returns

None.

### Example

```
Unix> calibredrv -m layout.oas
% set wb [find objects -class cwbWorkBench]
% layout0 layerFilters -add mb 1 50 60 100
% $wb viewedLayoutClbk layout0
% $wb showWithLayerFilterClbk mb
--> Layers 1, 50, 60, and 100 are filtered for viewing.
```

### Related Commands

None.

## \$cwb updateDisplay

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Redraws the display, the layer panel, and the cell hierarchy panel. The command clears rulers, selections, and contexts in the display. Use this command after using layout modification commands.

#### Note



The redraw action performed from this command differs from the GUI redraw action; the GUI redraw (**View > Update Display**) only redraws the display.

---

### Usage

**\$cwb updateDisplay**

### Arguments

None.

### Returns

None.

### Example

```
set wb [find objects -class cwbWorkBench]
$cwb updateDisplay
```

### Related Commands

None.

## \$cwb viewedLayoutClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

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

### Usage

**\$cwb viewedLayoutClbk** *layout\_handle*

### Arguments

- *layout\_handle*  
A required argument specifying the new layout to display.

### Returns

None.

### Example

```
set wb [find objects -class cwbWorkBench]  
$wb viewedLayoutClbk layout2  
$wb getSelectionRect dbunits  
$wb viewedLayoutClbk layout4  
$wb getSelectionRect dbunits
```

### Related Commands

None.

## **\$cwb zoomAllClbk**

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### **Description**

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

### **Usage**

**\$cwb zoomAllClbk**

### **Arguments**

None.

### **Returns**

None.

### **Example**

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

### **Related Commands**

[\\$cwb zoomToRectClbk](#)

## \$cwb zoomToRectClbk

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Zooms to the specified coordinates in the layout viewer.



#### 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. By default, coordinates are in database units (dbu).

### Returns

None.

### Example

None.

### Related Commands

[\\$cwb zoomAllClbk](#)

## Layout Object

The layout object manages all layout data loaded into the Calibre layout viewer 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**<sup>1</sup> 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.

### Usage

**layout all**

---

### Note



This is a great way to see what layouts are open in memory.

---

### Arguments

None.

### Returns

A list of all layout object handles.

---

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.

### Example

```
% layout all  
layout2 layout3 layout4
```

### Related Commands

None.

## 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.

---

### Usage

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

### Arguments

- *src*  
A required argument specifying the handle of the source layout object to be copied.
- *dest*  
An optional argument specifying 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*  
An optional argument specifying the name of the cell in the source layout to be copied. This cell becomes the topcell of the new layout. To copy the entire layout, specify the name of the topcell.
- *startDepth endDepth*  
An optional argument specifying the hierarchy depth range containing the data to be copied. Before 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*

An optional argument specifying the coordinates of the rectangular selection area. The first two values define the lower left corner, the second defines the upper right corner. When not specified, the entire layout is copied. By default, coordinates are in database units (dbu).

Coordinates can be specified as either an integer, an integer followed by the character ‘d’ for dbu, or a floating point number (or integer) followed by the character ‘u’ for microns.

- *selMode*

An optional switch that determines selection mode. Allowed values are 0 | 1 | 2 and contain | touch | clip.

- 0 (contain) — copies all geometry contained within the selection region.
- 1 (touch) — copies all geometry that touch or are contained within the selection region.
- 2 (clip) — clips geometries that are contained within the selection region. The results of this option leave no polygons or wires outside 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 preserve 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 preserve these properties.

- *-preserveHierarchy*

An optional flag specifying SREF- and AREF-instances enclosed in the selection rectangle are preserved. The default is to not preserve the hierarchy, creating a flat layout copy.

## Returns

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

## Examples

This example copies data to a new layout:

```
% layout all
layout2 layout3 layout4
% layout copy layout2 newlayout
newlayout
% layout all
newlayout layout2 layout3 layout4
```

This example copies to a new layout and sorts through the data:

```
% info nameofexecutable
/clbr_latest/ic/ixl/Mgc_home/pkgs/icwb/pvt/calibrewb
% layout copy layout0 myHandle lab3 0 0 -135.2u -23.6u -101.0u 2.4u
% lsort [myHandle layers]
1 4 7
% foreach layer [lsort [myHandle layers]] {
    foreach poly [lsort -dictionary \
        [myHandle iterator poly lab3 $layer range 0 end]] {
        puts "layer($layer) $poly"
    }
}
layer(1) -134000 -15000 -110000 -15000 -110000 -11000 -134000 -11000
layer(4) -114000 -12500 -105000 -12500 -105000 -11500 -114000 -11500
layer(7) -115000 -14000 -112000 -14000 -112000 -11000 -115000 -11000
layer(7) -132000 -16000 -127000 -16000 -127000 -11000 -132000 -11000
% layout delete myHandle
```

## Related Commands

[layout copy2](#)

## 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.

### Usage

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

### Arguments

- *src*  
A required argument specifying the handle of the layout object to be copied.
- *cell*  
A required argument specifying the name of the cell in the src layout to 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*  
A required argument specifying the coordinates of the rectangular selection area. The first two values define the lower left corner, the second defines the upper right corner. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- -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 preserve 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 preserve these properties.

### Returns

A system-generated handle for the layout object created as a result of the copy.

### Examples

This example copies data to a new layout:

```
% layout all
layout2 layout3 layout4
% layout copy2 layout2 a1220 0 0 200000 200000
```

layout5

This example copies to a new layout and sorts through the data:

```
% set L_copy2 [layout copy2 layout0 lab4 -135.2u -23.6u -101.0u 2.4u]
% lsort [$L_copy2 layers]
1 4 5 7
% foreach layer [lsort [$L_copy2 layers]] {
    foreach poly [lsort -dictionary [ \
        $L_copy2 iterator poly lab4 $layer range 0 end]] {
        puts "layer($layer) $poly"
    }
}
layer(1) -94000 12500 -104500 12500 -104500 -8500 -94000 -8500 -94000 \
-7500 -103500 -7500 -103500 11500 -94000 11500
layer(1) -134000 -15000 -110000 -15000 -110000 -11000 -134000 -11000
layer(4) -105000 17000 -121000 17000 -121000 -45000 -104000 -45000 \
-104000 -43000 -119000 -43000 -119000 15000 -105000 15000
layer(4) -108000 -14000 -81000 -14000 -81000 -10000 -108000 -10000
layer(4) -108000 0 -103000 0 -103000 3000 -108000 3000
layer(4) -114000 -12500 -105000 -12500 -105000 -11500 -114000 -11500
layer(4) -131000 -13000 -131000 -73000 -104000 -73000 -104000 -71000 \
-129000 -71000 -129000 -13000
layer(5) -108000 -16000 -81000 -16000 -81000 -1000 -108000 -1000
layer(7) -105000 1000 -103000 1000 -103000 2000 -105000 2000
layer(7) -115000 -14000 -112000 -14000 -112000 -11000 -115000 -11000
layer(7) -132000 -16000 -127000 -16000 -127000 -11000 -132000 -11000
% layout delete $L_copy2
```

## Related Commands

[layout copy](#)

## layout create

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Creates a layout object and returns a layout object handle. When no arguments are supplied, this command *creates* an empty layout object.

When a filename (or multiple filenames) are supplied, the command *loads* 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<sup>1</sup> in the application.

You can also use the command to create a new layout from an existing LEF/DEF, OpenAccess (OA), or Milkyway database.

### Usage

#### Syntax 1

```
layout create [-handle handle] [-dt_expand] [-preservePaths] [-ignoreGdsBoxes]
               [-preserveTextAttributes] [-preserveProperties]
```

#### Syntax 2

```
layout create [handle] fileName [-mapname Lname [LMAP]]... [{-map L D LMAP}...[-only]]
               [-dt_expand] [-preservePaths] [-ignoreGdsBoxes] [-preserveTextAttributes]
               [-preserveProperties] [-noReport] [-colorByDefinition] [-colorByPlacement]
               [-incr [sync]] [-handle handle] [-log logfile]
```

#### Syntax 3

```
layout create [handle] -files fileNameList [-mapname Lname [LMAP]]...
               [{-map L D LMAP}...[-only]] [-dt_expand] [-preservePaths] [-ignoreGdsBoxes]
               [-preserveTextAttributes] [-preserveProperties] [-noReport] [-colorByDefinition]
               [-colorByPlacement] [-incr [sync]] [-handle handle] [-log logfile]
```

#### Syntax 4

```
layout create {LEFDEF -def {def_file | @def_empty} -lef {lef_files}}
               {OA library cell view} | {MILKYWAY library cell view}
               [-dt_expand] [-preservePaths] [-preserveTextAttributes] [-preserveProperties]
```

### Arguments

- *handle*

An optional argument specifying 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 before this one. Thus, for the first layout created, the default handle is layout0.

---

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

- *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.

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.

- {-mapname *Lname LMAP*}...

Optional keywords and associated arguments used to map a layer name, *Lname*, in the GDS or OASIS file to a layer in the layout object, *LMAP*, when data is read into the new layout. *LMAP* is an optional integer layer number. If *LMAP* is omitted, the layout object takes the *Lname* number.

Each -mapname keyword maps one layer name to a specific layout layer. You can include multiple -mapname keywords to define more than one mapping.

All layers not specified in a -mapname statement are loaded and assigned the same layer number as in the GDS or OASIS file.

- {{-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.

- {LEFDEF -def {*def\_file* | @def\_empty} -lef {*lef\_files*}}

Optional keywords and associated arguments used to create a new layout from an existing LEF/DEF database.

- {OA *library cell view*}

Optional keywords and associated arguments used to create a new layout from an existing OpenAccess (OA) database. OpenAccess must be installed to access the OA database and read the data.

- {MILKYWAY *library cell view*}  
Optional keywords and associated arguments used to create a new layout from an existing Milkyway database. A Synopsys Milkyway license is required to access the Milkyway database and read the data.
- -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 preserve 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 preserve 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.
- -colorByDefinition  
An optional switch causing all layers within each jobdeck chip definition to be colored the same. The same color is assigned to both the level layers and their contained chip layers in the level and chip views. View modes can only be changed by closing the layout and reloading it with a different mode setting.

---

**Note**

Writing layouts from a database formatted for this type of viewing, will not result in an optimized layout. Databases should be loaded in Color By Chip mode when writing data in layout format.

---

- -colorByPlacement  
An optional switch that adds a visual distinction between unique jobdeck placement definitions. Each jobdeck placement is assigned a separate color. Using this switch causes

the jobToOasis command to generate unique chips in the OASIS file, which could result in large files; users of jobToOasis should leave the jobdeck in Color By Chip mode (the default).

---

**Note**



Writing layouts from a database formatted for this type of viewing, will not result in an optimized layout. Databases should be loaded in Color By Chip mode when writing data in layout format.

---

- `-handle handle`

An optional keyword used to indicate the handle name to assign to the newly created layout handle. It can be used instead of the optional *handle* in the first argument position. The last handle is used if both are entered or if there are multiple *-handle* arguments.

- `-incr [sync]`

An optional keyword used to invoke the layout viewer in incremental mode, which uses a cache file to aid the initial drawing speed. Invoking the layout viewer with this switch instructs the tool to load the layout incrementally and to generate or update a cache file if needed. When a layout is incrementally loaded, only hierarchical structures are loaded.

If the optional *sync* is supplied, the tool forces syncing the cache file (Peek Cache Repository, PCR) with the layout.

- `-log logfile`

An optional keyword used to write the information to a log file. Specify the file name using *logfile*. To print the contents to the terminal window, specify “stdout” for *logfile*. The `-log` option only works if the input file is GDS.

## Returns

The handle for the new layout object.

## Examples

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

This example only reads in layer 1.

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

This example reads in all layers and maps layer 1 to 101 and layer 2 to 102.

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

This example reads in all layers and maps layer 1, datatype 7 to layer 101.

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

The example only reads in layername LAYER101.

```
layout create f1.gds -mapname LAYER101 -only
```



This example reads in all layers, but maps layername LAYER101 to 101.

```
layout create f1.gds -mapname LAYER101 101
```

This example moves shapes from one layer to another without going through each shape. It maps a layer name to overcome two duplicate layer names for the layer A28.

```
set L1 [layout create mylayout.oas -mapname A28 900 -dt_expand]
$L1 layernames 900 A28dup
$L1 oasisout mgc.oas -map 900 1 0 -map 323.228 323 228
```

This example uses an incremental load cache file. This is useful in helping reduce loading time when reviewing Calibre results with very large layout files and mostly flat designs, such as the output from MDP tools.

```
layout create f1.gds -incr
```

## Related Commands

None.


## layout delete

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Deletes the layout specified by *handle*.

---

 **Caution** This command should only be used for deleting layouts when running scripts in batch mode without a GUI. If the script is working with a displayed layout in the GUI, use the [\\$cwb deleteLayoutClbk](#) command or errors will be generated.

---

### Usage

**layout delete** *handle*

### Arguments

- *handle*  
A required argument specifying the handle of the layout to be deleted.

### Returns

None.

### Examples

```
% layout all
layout2 layout3 layout4 layout5
% layout delete layout3
% layout all
layout2 layout4 layout5
```

### Related Commands

[\\$cwb deleteLayoutClbk](#)

## layout droasis

Tools Supported: Calibre WORKbench, LITHOview, MDPview

### Description

Loads a disk-resident OASIS file in read-only mode. A .fvi index file is automatically generated in the directory containing the file if one is not available at load time.

This command is not supported in Calibre DESIGNrev.



You can also load a disk-resident OASIS file from the batch command line by specifying it with the -v option.

---

### Usage

**layout droasis** *filename*

### Arguments

- *filename*

A required argument specifying the filename of the OASIS file to read in. The filename can be a relative or absolute pathname.

### Returns

The layout handle.

### Example

```
layout droasis hier.oas
```

### Related Commands

None.

## layout filemerge

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Performs a disk-based file merge on multiple GDS input files or multiple OASIS input files without loading them into memory. The input files must be exclusively GDS files or OASIS files, the two file types cannot be mixed. An import-style (one topcell is maintained) and merge-style (each file is given its own topcell) of the output is supported.

Input and output files must be in the same format. For example, GDS input files must output a GDS output file. OASIS input files must output an OASIS output file.

All of the input files must be of the same precision; no precision handling is performed.

Gzipped input and output files are supported by this command. Input files are expanded on disk at the location specified by the `-tmp` option. If `-tmp` is not specified, the command uses the `MGC_CWB_TMP_DIR` environment variable. If this variable is not set, the command expands files in the current directory. Output compression is driven by the `.gz|.GZ` file extension.

### Usage

- **layout filemerge -in *input\_file1* [*layer\_bump1*]**  
**-in *input\_file2* [*layer\_bump2*]**  
**[-in *input\_file3* [*layer\_bump3*] . . .]**  
**-out *output\_file***  
**[-append | -overwrite | -rename | -force\_rename]**  
**[-smartdiff]**  
**[-exclude\_layer *layer1 layer2 ... layerN*]**  
**[-include\_layer *layer1 layer2 ... layerN*]**  
**[-createtop *cellname*]**  
**[-topcell *topcellname*]**  
**[-map\_cell *cellname mapcellname*]**  
**[-preserve *filterfile*]**  
**[-tmp *tmpdir*]**  
**[-verbose]**

### Arguments

- **-in *input\_file1* [*layer\_bump1*]**  
A required option specifying an input file name. At least two file names must be specified. If you specify a *layer\_bump* value, all of the layers in that file will have their layer number incremented by that value.
- **-out *output\_file***  
A required option that specifies an output file.
- **-append | -overwrite | -rename | -force\_rename**  
Optional switches that specify how cells that have the same name in the input files are handled:

- -append - Concatenates the two cells contents together. This is the default.

---

**Note**



If the -smartdiff option is set, and a name conflict & cell object or properties are different, cells are appended. Otherwise, no append takes place if there is only a name conflict.

---

- -overwrite - Writes the second file cell contents over the contents of the first, deleting the contents of the first file's cell.

---

**Note**



If the -smartdiff option is set, and a name conflict & cell object or properties are different, cells are overwritten. Otherwise, no overwrite takes place if there is only a name conflict.

---

- -rename - Renames both of the duplicate cells, to make them unique.

---

**Note**



If the -smartdiff option is set, and a name conflict & cell object or properties are different, cells are renamed. Otherwise, no rename takes place if there is only a name conflict.

---

- -force\_rename - Renames all cell names to be unique.
- -smartdiff  
 If this optional switch is set, additional processing occurs when cellname conflicts occur. In the case of a conflict, the command compares topologies in the cells, and it only renames cells when the cell contents are different. This option provides you with a cleaner hierarchy for the output file, and it minimizes the file size.
- -exclude\_layer *layer1 layer2 ... layerN*  
 If this optional switch is specified, the output file only receives objects not on the specified layers and optional datatypes. The -exclude\_layer and -include\_layer options are mutually exclusive.
- -include\_layer *layer1 layer2 ... layerN*  
 If this optional switch is specified, the output file only receives objects on the specified layers and optional datatypes. The -exclude\_layer and -include\_layer options are mutually exclusive.
- -createtop *cellname*  
 An optional switch that, if specified, runs the command in merge mode. A topcell is created with the supplied name and each input file is created with its topcell being a child of the new topcell. If this switch is not specified, the input layouts are merged in import-style and no top cell will be created.
- -topcell *topcellname*  
 An optional switch that sets *topcellname* to the topcell of the merge result layout. This is especially useful in removing unreferenced cells from an IP library file.

- `-map_cell cellname mapcellname`

An optional switch that renames cell name to mapped cell name. The *cellname* parameter accepts the '\*' wildcard. This wildcard will find zero or more cell occurrences, and using it in a search such as *ca\*t* would find *cat*, *cart*, and *celebrant*.

- `-preserve filterfile`

An optional priority cell list switch used to preserve cells during merging, which is useful for enabling assembly of library files. The cells are specified in the *filterfile*, where each priority file and cell is defined on a line as a pair value:

```
filename_x cellname_a
filename_x cellname_b
filename_z cellname_c
```

If a cell name is specified more than once, with different file names, the last pair value defined will take precedence. The priority cell list applies to all modes in this command.

---

### Caution



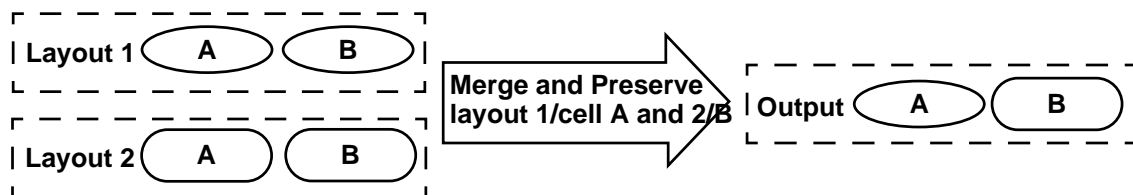
If this switch is not used, and if a cell name conflict is discovered, the last input file in the command line will overwrite cells in other input files.

---

The *-preserve* option makes controlled merges possible. When there is a cell name conflict in filemerge overwrite mode, the last input file on the command line will overwrite cells from other input files. However, when more than one library file presents input, some cells may not come from the last file in the command line. For example, if you want to use library cell A from layout 1 and library cell B from layout 2, and both layout 1 and layout 2 have a cell A and cell B. Without the *-preserve* option, you could only have A and B in the output from the same layout. The *-preserve* option allows you to specify cell A, layout 1 and cell B, layout 2, to preserve them in the final result.

The *-preserve* option ensures a cell/file pair gets preserved in the final merging result. In another words, the *cell* in that specific *file* always overwrites other cells with the same name from other files in the filemerge process.

**Figure 6-2. Merge and Preserve**



- `-tmp tmpdir`

An optional switch and directory name that allows you to specify the location of gzip temporary expanded files for further indexing in *-append* or *-rename* mode, useful for performance reasons. If you do not specify a directory, the file merge uses the

MGC\_CWB\_TMP\_DIR environment variable, if set. If it is not set, it uses the current directory.

- -verbose

An optional switch that reports merge conflicts. Information provides indicates cells causing name conflicts as well as actions taken for each conflict.

## Returns

The merged file is written to the filename specified by *output\_file*.

## Example 1

Combines the files “adder4\_2.gds” and “adder4\_1.gds” into the output file “toto\_import\_rename.gds.” Any cells that have the same name are uniquely renamed.

The result is an import-style merging of the two layouts.

```
layout filemerge -in adder4_2.gds -in adder4_1.gds -out  
toto_import_rename.gds -mode rename
```

## Example 2

Combines the files “adder4\_2.gds” and “adder4\_1.gds” into the output file “toto\_rename60.gds.” All the layers in “adder4\_2.gds” are incremented by 60. Any cells that have the same name are uniquely renamed.

The result is a merge-style merging of the layouts; the output file has a topcell named TOP.

```
layout filemerge -in adder4_2.gds 60 -in adder4_1.gds -out  
toto_rename60.gds -rename -createtop TOP
```

## Example 3

Combines the files “adder4\_2.gds” and “adder4\_1.gds” into the output file “toto\_force\_rename.gds.” All cells are uniquely renamed.

The result is a merge-style merging of the layouts; the output file has a topcell named TOP.

```
layout filemerge -in adder4_2.gds -in adder4_1.gds -out  
toto_force_rename.gds -force_rename -createtop TOP
```

## Example 4

Combines the files “adder4\_2.gds” and “adder4\_1.gds” into the output file “toto\_force\_rename.gds.” All the layers in “adder4\_2.gds” are incremented by 60. All cells are renamed uniquely.

The result is a merge-style merging of the layouts.

```
layout filemerge -in adder4_2.gds 60 -in adder4_1.gds -out  
toto_force_rename60.gds -force_rename -createtop TOP
```

## Example 5

Examples of combining files using layer filtering.

```
layout filemerge -in probemod.oas -in probemod2.oas -include_layer 8 58 \
```

```
-out probemod_filemerge_overwrite_include.oas -overwrite  
layout filemerge -in probemod.oas -in probemod2.oas -exclude_layer 8 58 \  
-out probemod_filemerge_overwrite_exclude.oas -overwrite
```

## Related Commands

None.



## 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 or combined according to the mode you select.

### 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**  
A required argument specifying the handle or filename of the first layout to be merged.
- **handle2** | **file2**  
A required argument specifying the handle or filename of the second layout to be merged.
- **bump**  
A required argument defining an integer value added to the layer numbers of each layer in the second layout (**handle2** or **file2**). 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 layoutThe 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 (adds `_WBx` 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 (adds `_WBx` 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*.

---

**Note**



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 record attribute data when reading data into or writing data from the application. The default is to preserve 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 and PROPVALUE). The default is to preserve these properties.

---

**Note**



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 to not to automatically align merged 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 depending on the types of layout files Calibre accepts:

- **GDS files** — They 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** — They 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.

If you are merging two OASIS files *or* a GDS file in microns with an OASIS file, no re-scaling is required, due to the two layouts having the same database unit. OASIS files always use microns for units, and therefore they have the same scale.

---

### Note



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 layout to that value. For best results, Mentor Graphics recommends you define the physical size of the coordinate units of both layouts before merging them using the [\\$L units database](#), [\\$L units user](#), and [\\$L units microns](#) commands.

---

For example, changing a GDS layout from precision 1000 to 2000, requires a change of the database units from  $1e^{-9}$  to  $5e^{-10}$ , the user units value from .001 to .0005, and the layout to be scaled by 2. The scale changes an edge of say 1 dbu to 2 dbu, while maintaining the same physical size:

- File\_precision1000.gds:

```
$L units database = 1e-9
$L units user = 0.001
$L units microns = 1000    (this is the precision)
```
- File\_precision2000.gds:

```
$L units database = 5e-10
$L units user = 0.0005
$L units microns = 2000
```
- Merged files yields:

```
$L units database = 5e-10
$L units user = 0.0005
$L units microns = 2000
```

## Returns

The new layout handle.

## Example

```
% layout all
layout2 layout4 layout6
% layout merge mhandle layout2 layout4 1000
mhandle
```

## Related Commands

None.

## layout overlays

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays all defined overlay handles.

### Usage

**layout overlays**

### Arguments

None.

### Returns

Returns all defined overlay handles.

### Example

```
% layout overlays  
overlay0
```

### Related Commands

[\\$O layoutHandle](#)

[\\$O overlayCells](#)

[\\$L isOverlay](#)

[\\$L layerFilters](#)

[\\$O layouts](#)

[\\$O overlayout](#)

[\\$L isReferenced](#)

## layout peek

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Creates a peek object which displays units, precision, list of cells, list of topcells, list of layers, and other information without loading a layout file into memory.

### Usage

**layout peek** *fileName* [-bbox *cellname*] [-cblock] [-cblockcount] [-cellcount] [-cells]  
[-child *cellname*] [-lastmodtime] [-layers] [-parent *cellname*] [-precision]  
[-refcount *cellname*] [-topcell] [-topcells] [-type] [-undefcells] [-units]  
[-handle [*handlename*]] [-cache *directory*]

---

#### Note



Only one -cblock or -cblockcount option can be defined in a layout peek command. If more are defined, the later option will take precedence.

---

### Arguments

- *fileName*

A required argument specifying 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.

- -bbox | -cblock | -cblockcount | -cellcount | -cells | -child | -lastmodtime | -layers | -parent | -precision | -refcount | -topcell | -topcells | -type | -undefcells | -units

Specifies information to query. If no option is specified, units are displayed.

- bbox — Queries cell's coordinate space for the specified *cellname*.
- cblock — Queries for cblock existence.
- cblockcount — The number of cblocks in this file.
- cellcount — Queries the number of cells in layout.
- cells — All cell names available in layout.
- child — A list of children for the specified *cellname*.
- lastmodtime — Queries for creation date from GDS file.

---

#### Note



The command only accepts GDS files.

---

- layers — All layers available in layout.
- parent — The parent cell for the specified *cellname*.

- precision — A value of 1/units.
  - refcount — A count of reference instances to the specified *cellname*.
  - topcell — Determines the topcell with the most descendants.
  - topcells — All topcells available in layout.
  - type — Specifies the file type, such as GDS, OASIS, or OASIS-STRICT-MODE (for OASIS files in strict mode).
  - undefcells — Displays cells referenced but not defined.
  - units — Provides units in microns.
- -handle *handlename*  
An optional keyword used to indicate the handle to assign to the newly created peek handle. Multiple **layout peek** queries can use the -handle option to avoid rereading layout files to extract the peek information, by invoking the handle as “*handlename peek ...*”
  - -cache *directory*  
An optional keyword which determines the location of a cache file to prevent rereading the layout to retrieve this information. Invoking the command with this option instructs the tool to locate the cache file to generate or update data as needed.  
  
The presence of the -handle option causes the command to use the cache. The cache is always used by the -handle option, and the cache is found in the default location if the -cache option is not used. If the -cache *directory* information is not specified, the command will use the MGC\_CWB\_PCR\_PATH variable. If this variable is not set, the command will write the cache file to the current directory.  
  
For multiple queries, the **layout peek** command rereads the layout to extract information. To improve performance, the ability to cache information is provided for with the -cache option.

## Returns

Returns units, precision, list of cells, list of topcells, list of layers, and other information without loading a layout file into memory.

- **-bbox** — Returns cell's coordinate space as  $\{x\ y\ width\ height\}$  for the specified *cellname*. X and Y are the coordinates of the lower left corner of the bounding box. The width and height of the bounding box are defined in um.
- **-cblock** — Returns 1 to indicate existence of one or more cblocks, and 0 otherwise.
- **-cblockcount** — Returns the number of cblocks in the layout.
- **-cellcount** — Returns the number of cells in the layout.
- **-cells** — Returns all cell names available in the layout.
- **-child** — Returns a list of children for the specified *cellname*.
- **-lastmodtime** — Returns creation date from the specified GDS file.
- **-layers** — Returns all layers available in layout, including layers which are defined yet contain no shapes.
- **-parent** — Returns the parent cell for the specified *cellname*.
- **-precision** — Returns a value of 1/units.
- **-refcount** — Returns a count of reference instances to the specified *cellname*.
- **-topcell** — Returns the topcell with the most descendants.
- **-topcells** — Returns all topcells available in the layout.
- **-type** — Returns the file type.
- **-undefcells** — Returns cells referenced but not defined.
- **-units** — Returns the units in microns.

If no option is specified, the units are returned.

## Examples

Issuing the following commands from inside the Calibre layout viewer shell results in the listed output:

```
% layout peek mix.gds -bbox r3500
{r3500 {-2000 -2000 4000 4000}}

% layout peek mix.gds -cblock
1

% layout peek mix.gds -cblockcount
2

% layout peek mix.gds -cellcount
9

% layout peek mix.gds -cells
r1220 r1230 r1240 r1310 r1620 r1720 r2310 r3500 mix

% layout peek mix.gds -child mix
{mix {r3500 r1620 r1220 r2310 r1720 r1230 r1240 r1310}}
% layout peek mix.gds -child r3500
{r3500 {}}
```



```
% layout peek mix.gds -lastmodetime
3-6-2007 19:49:52

% layout peek mix.gds -lastmodetime -precision
date {3-6-2007 19:49:52} precision 1000.0

% layout peek mix.oas -lastmodetime -precision
date {Date not available} precision 1000.0

% layout peek mix.gds -layers
{1 0} {2 0} {4 0} {5 0} {6 0} {7 0} {8 0} {9 0} {10 0} {28 0} {29 0} {30 0}

% layout peek mix.gds -parent r3500
{r3500 mix}
% layout peek mix.gds -parent mix
{mix {}}

% layout peek mix.gds -precision
1000.0

% layout peek mix.gds -refcount r3500
{r3500 122}
% layout peek mix.gds -refcount mix
{mix 0}

% layout peek mix.gds -topcell
mix

% layout peek mix.gds -topcells
mix add1 add2

% layout peek mix.gds -type
GDS

% layout peek mix.gds -undefcells
% r1230

% layout peek mix.gds -units
0.001
```

To use this command to create a *peek* handle, which is much like a layout handle, and to make use of a cache:

```
set layout [lindex layout_file.gds]
puts "Generating cache file for $layout."

# Create peek handle and use cache.
set peek [layout peek $layout -handle mypeek -cache . ]

# Read out data and dump to shell.
array set peekdata [$peek peek -type -precision -layers -bbox -topcells \
-cellcount]
puts "Format: [string tolower $peekdata(type)]"
puts "Precision: [expr {$peekdata(precision) * 1.0}]"
puts "Topcells: [lsort -dictionary $peekdata(topcells)]"
puts "Cell count: $peekdata(cellcount)"
```

```
puts "Layers: $peekdata(layers) "  
  
# Get bounding box for topcells.  
set bboxes [$peek peek -bbox [lsort -dictionary $peekdata(topcells)]]  
foreach topbbox $bboxes {  
    puts "boundingbox for topcell [lindex $topbbox 0]: [lindex $topbbox 1]"  
}  
  
# Delete the peek handle, preserving cache for subsequent runs.  
$peek delete -preserve cache
```

To use this command from the UNIX prompt, use the following configuration:

```
% calibredrv -a puts [layout peek mix.gds -cells -precision] >> out.txt
```

To set an alias at the UNIX prompt, use the following alias command:

```
% alias layers 'calibredrv -a "puts [layout peek \!* -layers]"'  
  
% layers lab2.gds  
{1 0} {2 0} {4 0} {5 0} {6 0} {7 0} {8 0} {9 0} {10 0} {28 0} {29 0} {30 0}
```

## Related Commands

[\\$P delete](#)

[\\$P file](#)

[\\$P peek](#)

## 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.



#### Caution

**This is a deprecated command that is no longer necessary.** When loading a layout with the **layout create** command and subsequently viewing the layout with the DESIGNrev GUI, in previous releases all layer colors defaulted to yellow. The proper GUI default colors are now assigned when the layer objects are created, without requiring you to load unique layer colors by hand.

---

### Usage

**layoutSetDefaultColors** *layout\_handle*

### Arguments

- *layout\_handle*

A required argument specifying the layout handle of a currently loaded layout.

### Returns

None.

### Example

None.

### Related Commands

None.

## layout tbmode

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Sets how text extents are handled.

### Usage

**layout tbmode** [classic | origin | extent | ignore]

### Arguments

- classic | origin | extent | ignore

An optional argument specifying the mode to be used.

- classic — This option is selected by default. The text origin is used as the lower-left coordinate of the extent and the height is set fixed to .25 microns independent of the actual text attributes or the user units set for the design. The contribution of the width is zero no matter how many characters the text object has. The text-extent is added to the cell-extent.
- origin — This option sets the text origin in the lower-left and upper-right coordinate of the extent. The origin of the text is added to the cell-extent.
- extent — This option sets the text extent to be database driven by the height of the text object. When choosing the Extent option, cell-bounding calculation behavior is similar to that of other major layout viewers. The text-width is calculated by using the character length multiplied by a fixed ratio of the height. This ratio is given by the font geometry used within the application, and is currently set to 7/10.

The under-stroke of certain characters such as “g” may cause the lower-left coordinate to be slightly lower than the origin of the text. If the text is rotated or aligned this is reflected in the extent calculation. The text-extent is added to the cell-extent.

Choosing the Extent option will affect how text objects are drawn by the application. With all other styles you have the option of setting either fixed or scaled drawing, but this is not the case when choosing Extent.

When Extent is chosen, text objects are not drawn using the scaling and default height options normally available in the Sessions tab. Text is drawn exactly as specified in the layout database.

- ignore — Text does not contribute to the cell-extent in any way.

### Returns

The new handling mode, or the current handling mode if no arguments were supplied.

## Example

```
layout tbmode origin
```

## Related Commands

None.

## \$L allowupdates

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Resumes cell bounding box (bbox) calculations. Bbox calculations are always performed for operations that modify the database. For mass edits, you can postpone and resume bbox updates:

- The **\$L holdupdates** command holds bbox updates until the end of the Tcl procedure that the command was called in. Cascaded calls are allowed, and updates are held back until the end of the procedure of the first call to **\$L holdupdates**.
- The **\$L allowupdates** command is not usually needed. It is useful for triggering a bbox calculation update immediately, instead of waiting for the automatic trigger at the end of the procedure.

If **\$L holdupdates** is called, it automatically calls **\$L allowupdates** at the end of a Tcl procedure. Calling the **\$L allowupdates** directly gives you a way to allow updates earlier.

#### Note



If bbox updates are held by the **\$L holdupdates** command, any queries that return the bbox of the layout could be potentially incorrect as changes which modify the bbox of cells have not yet been executed.

#### Caution



**\$L allowupdates** can *only* be called from within a Tcl procedure. It cannot be called from the command line.

### Usage

**\$L allowupdates**

### Arguments

None.

### Returns

None.

### Example

In this example, the `create_layout_and_add_shapes_sample1` procedure calls the `create_many_shapes` procedure, which *holds* bbox calculations. After the end of this procedure, bbox calculation is triggered, and queries to get the cell bbox are now correct.

The `create_layout_and_add_shapes_sample2` procedure suspends bbox calculations until the call to allow updates. Any query of the bbox is wrong until the trigger of the bbox calculation with `allowupdates`.

```
proc create_layout_and_add_shapes_sample1 {} {  
    set lay [layout create]  
    $lay create cell TOP  
  
    create_many_shapes $lay TOP  
  
    # Returns correct bbox  
    set bbox [$lay bbox TOP]  
}  
  
proc create_many_shapes { lay cell } {  
    $L holdupdates  
  
    $lay create polygon $cell ...  
    $lay create polygon $cell ...  
    $lay create polygon $cell ...  
}  
  
proc create_layout_and_add_shapes_sample2 {} {  
    set lay [layout create]  
    $lay create cell TOP  
  
    $L holdupdates  
  
    create_many_shapes $lay TOP  
  
    # Returns wrong bbox!  
    set bbox [$lay bbox TOP]  
  
    $L allowupdates  
  
    # Returns correct bbox  
    set bbox [$lay bbox TOP]  
}
```

## Related Commands

[\\$L holdupdates](#)

## \$L ancestors

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays unique ancestors of the specified cell. An ancestor is any cell which has the specified cell in it's sub-hierarchy. For example, if cell A references cell B which in turn references cell C, executing the command with a *cellName* of C will return both A and B.

### Usage

**\$L ancestors** *cellName*

### Arguments

- *cellName*  
A required argument specifying the name of the cell whose ancestors are to be returned.

### Returns

Unique ancestors of the specified cell.

### Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout4
% $lay ancestors y3120
m1000
```

### Related Commands

[\\$L children](#)



## \$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***  
A required argument specifying the name of the first layer to be ANDed.
- ***L2in***  
A required argument specifying the name of the second layer to be ANDed.
- ***Lout***  
A required argument specifying the name of the output layer for the operation.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay AND 2 4 100
% $wb updateDisplay
```

### Related Commands

[\\$L NOT](#)

[\\$L OR](#)

[\\$L XOR](#)

## \$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 rule check names.
- All text objects are written as rule check text lines.
- If more than one cell is present, it will result in a user error.

#### Note



The layout type must be converted to ASCII before you can use this command. Use the [\\$L layoutType](#) command to convert the layout.

---

### Usage

**\$L asciiout** *fileName*

### Arguments

- *fileName*

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

### Returns

None.

### Examples

Writes out an RDB file as an ASCII file:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
$lay asciiout out.results
```

Converts an OASIS file to an ASCII text file:

```
Unix> calibredrv -shell
% set lay [layout create mylay.oas -dt_expand -noReport -
preserveProperties]
% $lay layoutType set ascii
% $lay asciiout mylay.ascii
```

### Related Commands

[\\$L instancedbout](#)

## \$L bbox

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays 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]

**\$L bbox *cellName*** [-s\_bounding\_box] (OASIS files only)

**\$L bbox -p44\_chip\_window** (OASIS files only)

### 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.
- **-s\_bounding\_box**  
An optional argument that when present, instructs the software to retrieve the S\_BOUNDING\_BOX property value (for viewer compatibility with the OASIS.MASK standard). This option is only available for OASIS-format files.
- **-p44\_chip\_window**  
An optional argument that when present, instructs the software to retrieve the P44\_CHIP\_WINDOW property value (for viewer compatibility with the OASIS.MASK standard). This option is only available for OASIS-format files.

### Returns

Returns the bounding box of the specified cell, in database units.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay bbox a1620
-4250 0 32250 85000
```

### Related Commands

None.

## \$L cellname

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Renames the specified cell.

### Usage

**\$L cellname *oldName newName***

### Arguments

- *oldName*  
A required argument specifying the current name of the cell.
- *newName*  
A required argument specifying the new name for the cell.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0  
% set lay [$wb cget -_layout]  
layout1  
$lay cellname a1620 a1620r  
$wb updateDisplay
```

### Related Commands

[\\$L cells](#)

[\\$L children](#)

## \$L cells

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays a list of all the cells in the layout \$L.

### Usage

**\$L cells**

### Arguments

None.

### Returns

A list of all the cells in the layout \$L.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay cells
lab1 a1220 a9500 a1720 a1230 a1310
```

### Related Commands

[\\$L cellname](#)

[\\$L children](#)

## \$L children

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays 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*

A required argument specifying the name of the cell whose children are to be returned.

### Returns

Unique children, but not subsequent descendants, of the specified cell.

### Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay cells
lab4 a1000 a2000 a3000
% $lay children lab4
a1000 a2000 a3000
% $lay children a2000
%
```

### Related Commands

[\\$L ancestors](#)

[\\$L cellname](#)

[\\$L cells](#)

## \$L clips

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Retrieve the set of clips for a particular cell.

### Usage

**\$L clips**  
*cellName*

### Arguments

- **cellName**  
Name of the cell.

### Returns

A Tcl list of Tcl lists of the attributes for the constituent clips.

### Example

```
set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
set lay [$wb cget -_layout]
layout0
$lay clips a1220
{1 {Clip 1} 1716 52549 16921 20622 true false}
```

### Related Commands

[\\$L clipsout](#)

[\\$L create clip](#)

[\\$L delete clip](#)

## \$L clipsout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Writes the currently loaded clip markers to a file which can be loaded with the [\\$L create clip](#) command in a separate, interactive instance of the layout viewer.

### Usage

**\$L clipsout**  
*fileName*  
*cellName*

### Arguments

- **fileName**  
Name of the file to write.
- **cellName**  
Name of the cell.

### Returns

None.

### Examples

This example specifies a clip filename with the \$L clipsout command:

```
# script1.tcl - clipsout example
set L [layout create lab1.gds]
$L create clip lab1 -name {Clip 10} -mark -clip {36000 250000
359000 90000}
$L clipsout lab1.clips lab1
```

The following example script creates two clip markers, one with a zero width & height, and the other with non-zero width & height (perhaps showing a region to inspect). It then writes the clip markers to a file that can be loaded in a separate, interactive instance of the viewer:

```
# script2.tcl - a sample script
layout create lab2.gds
set lay [lindex [layout all] 0]
set topcell [$lay topcell]
$lay create clip $topcell -name {Clip 1} -mark -clip {1000000 5000000 \
1000000 5000000}
$lay create clip $topcell -name {Clip 2} -mark -clip {1000000 4000000 \
2000000 5000000}
$lay clipsout lab2.clips $topcell
```

Here is an example of how the script could be used:

```
$ calibredrv -shell script.tcl
$ calibredrv -m lab2.gds -incr lab2.clips
```



## Related Commands

[\\$L clips](#)

[\\$L create clip](#)

[\\$L delete clip](#)

## \$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.

You can use a mixture of layer numbers and layer names to define net extraction connect statements. If a layer name cannot be identified, or if a layer name matches multiple layer numbers, an error will be generated. Internally, layer names are translated into layer numbers, so if layer names are used to create \$L connect statements, a query on \$L connect commands will return layer *numbers*. Layer names must be loaded before using them in a \$L connect statement.

### Caution



The \$L connect and \$L disconnect statements ignore ordering of the *layer1* and *layer2* layers. The following commands are considered identical and will result in only one connection:

\$L connect 1 2

\$L connect 2 1

In previous releases, the two statements were considered distinct.

Command can be used on overlays. Existing layer connections in any overlaid layouts are copied into the overlay, and then new connections can be added across the layouts' layers.

### Usage

**\$L connect** *layer1 layer2* [by *layer3*] [via *viacell*] [width *width*]

### Arguments


- *layer1*  
The first layer to connect.
- *layer2*  
The second layer to connect.
- by *layer3*  
An optional third layer used to contain shapes that connect the two layers.
- via *viacell*  
An optional layer used to specify the cell name of a via cell.
- width *width*  
An optional size used to specify the width of the box dropped on the layer to connect different segments of a multi-layer path. By default, the width is in database units (dbu).

Coordinate lengths can be specified as either an integer, an integer followed by the character ‘d’ for dbu, or a floating point number (or integer) followed by the character ‘u’ for microns.

## Returns

The command supports returning existing connection definitions for a specific layer. In this case, the command will return layers connected to the specified layer. To distinguish a via layer, a non-via layer will always list itself in the list of layers it connects to.

---

 **Note** Duplicate layer numbers are possible.

---

## Example

Here are several \$L connect statements.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay connect 1 3
% $lay connect 2 4 by 8 width 10.
% $lay connect 10 12 via 16.
```

## Related Commands

[\\$L disconnect](#)

## \$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; all cells on the specified layer are copied to the new layer.

### Usage

**\$L COPY *Lin Lout***

### Arguments

- ***Lin***  
A required argument specifying the layer number or *layer.datatype* designation for the layer to be copied.
- ***Lout***  
A required argument specifying the layer number or *layer.datatype* designation for the layer that is a copy of ***Lin***.



This command cannot be used to copy a single layer's cell contents to another layer. Instead, use the [\\$L iterator \(poly | wire | text\)](#) command.

---

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay COPY 2 200
% $wb updateDisplay
```

### Related Commands

[\\$L COPYCELL GEOM](#)

[\\$L iterator \(poly | wire | text\)](#)

## \$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***  
A required argument specifying the handle of the layout containing the geometry to copy.
- ***srcCell***  
A required argument specifying the name of the cell containing the geometry to copy.
- ***srcLayer***  
A required argument specifying the layer number or *layer.datatype\_number* designation for the layer containing the geometry to copy.
- ***destCell***  
A required argument specifying the The name of the cell into which the geometry will be copied.
- ***destLayer***  
A required argument specifying the layer number or *layer.datatype\_number* designation for the layer onto which the geometry will be copied.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay COPYCELL GEOM
$lay a1220 28 a1230 280
% $wb updateDisplay
```

### Related Commands

[\\$L COPY](#)

## \$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***  
A required argument specifying the name of the cell to be created. The name cannot conflict with an existing cell name.
- ***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.

### Returns

None.

### Example

Extract cell from an existing layout into a new GDS file:

```
# Load existing layout.
set Llayout [layout create mylayout.gds]

# Create an empty layout.
set Lnew [layout create]

# Create topcell for new layout.
$Lnew create cell TOP

# Copy MyGeom cell to new layout, calling it MyGeomNew. Copies all
# geometry, but not placements.
$Lnew create cell MyGeomNew $Llayout MyGeom

# Place MyGeomNew cell reference in new layout.
$Lnew create ref TOP MyGeomNew 0 0 0 0 1
```

```
# Export new layout.  
$Lnew gdsout new.gds
```

## Related Commands

[\\$L create layer](#)

[\\$L create ref](#)

[\\$L create wire](#)

[\\$L create polygon](#)

[\\$L create text](#)

## \$L create clip

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Supports the creation of clips, of which markers are one type. The statement allows you to add markers with conventional zero width and height, and also with non-zero width and height to highlight a region for inspection. You can also load clips from an RDB file.

The [\\$L clipsout](#) command can be used to write the currently loaded clips to a file.

### Usage

#### **\$L create clip**

```
cellName  
[-layer {layer | *}]  
[-depth depth]  
[-name name]  
[-incr]  
[-mark]  
{  
    [-halo microns]  
    [-clip {llx lly urx ury}]  
    [-clipsFile fileName]  
    [-layout {handle | *} {cellName | *} layer]  
    [-rdb rdbFile {ruleCheck | *}]  
} ...
```

### Arguments

- **cellName**  
A required argument specifying the name of the cell.
- -layer {*layer* | \*}  
One layer or all layers. The switch defaults to the layout viewer.
- -depth *depth*  
Layout hierarchy *depth*. The switch defaults to the layout viewer.
- -name *name*  
The name of the clip.
- -incr  
Incrementally load the area of the clip, if the layout is incrementally loaded.
- -mark  
Place a *marker* dot in the center of the clip when the clip is drawn.



- **-halo *microns***  
Halo around polygons, in microns.
- **-clip {*llx lly urx ury*}**  
Load one area with user-supplied coordinates, lower left X & Y and upper right X & Y. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character ‘d’ for dbu, or a floating point number (or integer) followed by the character ‘u’ for microns.
- **-clipsFile *fileName***  
Load a clips file containing areas of user-supplied coordinates.
- **-layout {*handle* | \*} {*cellName* | \*} *layer***  
Load a layout.
- **-rdb *rdbFile* {*ruleCheck* | \*} }**  
Load RDB results.

## Returns

A list of IDs of each created clip.

## Examples

The following example script creates two clip markers, one with a zero width and height, and one with non-zero width and height (perhaps showing a region to inspect). It then writes these clip markers to a file that can be loaded in a separate, interactive instance of the layout viewer.

```
# script.tcl - a sample script
#
# This is how the script could be used:
# $ calibredrv -shell script.tcl
# $ calibredrv -m lab1.gds -incr lab1.clips
#
layout create lab1.gds
set lay [lindex [layout all] 0]
set topcell [$lay topcell]
$lay create clip $topcell -name {Clip 1} -mark -clip {1000000 5000000 \
1000000 5000000}
$lay create clip $topcell -name {Clip 2} -mark -clip {1000000 4000000 \
2000000 5000000}
$lay clipsout -fileName lab1.clips -cellname $topcell
```

The following example loads all “lab1” cell clips from the results in an RDB file. The following operates on the cell “lab1”, loads the RDB file “lab1.drc.results”, and uses a halo of 2 um.

```
$lay create clip lab1 -rdb lab1.drc.results * -halo 2
```

This example loads the “a1220” cell, and includes all layers.

```
$lay create clip a1220
```

This example only loads the results from the one check called “internal” in “lab1” cell.

```
$lay create clip lab1 -rdb lab1.drc.results internal -halo 2
```

This example loads layer 8.

```
$lay create clip lab1 -layer 8
```

This example loads all layers.

```
$lay create clip lab1 -layer *
```

This example loads a selection area.

```
$lay create clip lab1 -clip 10000.0 300000.0 140000.0 400000.0
```

This example names the new clip, “my clip 3”.

```
$lay create clip a1220 -name {my clip 3}
```

This example shows the command usage.

```
$lay create clip
```

## Related Commands

[\\$L clips](#)

[\\$L clipsout](#)

[\\$L delete clip](#)

## \$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*.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay create layer 420
% $wb updateDisplay
```

### Related Commands

[\\$L create cell](#)

[\\$L create ref](#)

[\\$L create wire](#)

[\\$L create polygon](#)

[\\$L create text](#)

## \$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***  
A required argument specifying the name of the cell in which the polygon will be created.
- ***layer***  
A required argument specifying the layer number or <layer number>.<datatype number> designation for the layer on which to create the polygon.
- **-prop *attr string* [G|U]**  
An optional flag plus associated values defining polygon property data (PROPATTR & PROPVVALUE). 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.  
  
G (GDS) is 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.
- ***x1 y1 x2 y2 .... x\_n y\_n***  
The coordinates of the vertices of the polygon. If only two coordinates are supplied, the software interprets the polygon as a box. At least two coordinates are required. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.

### Returns

None.

### Examples

This example creates and queries a polygon object:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
```

```
% set L [$cwb cget -_layout]
% $L create polygon lab5 11 10u 10u 10u 20.5u 20.5u 20.5u 10000 10000
% $L query polygon lab5 0 0 20.5u 20.5u
{11 10000 10000 10000 20500 20500 20500 10000 10000} 4 v 0.0 {20500 20500}
```

This example creates a polygon:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set ll [$wb cget -_layout]
layout0
% $ll create cell a2200
% $ll create layer 420
% $wb updateDisplay
Select a new cell
% $ll create polygon a2200 420 10 10 80 160
% $wb zoomAllClbk
```

## Related Commands

[\\$L create cell](#)

[\\$L create ref](#)

[\\$L create wire](#)

[\\$L create layer](#)

[\\$L create text](#)

[\\$L delete polygon](#)

## \$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***  
A required argument specifying the name of the cell that is the parent cell of the new cell reference.
- ***refcell***  
A required argument specifying the name of the cell that is to be referenced.
- ***x y***  
A required argument specifying the coordinates at which to place the origin of the cell. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- ***mirror***  
A required argument specifying the mirror to apply to the cell reference. Allowed values are 0 | 1.
  - 0 — do not apply a mirror
  - 1 — mirror across the x axis of the cell
- ***angle***  
A required argument specifying 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.
- ***mag***  
A required argument specifying the magnification to apply to the cell reference. By default, this is set to 1, which equates to no magnification. The value can be any real number greater than 0.
- ***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. 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.

G (GDS) is 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.

- `-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.

## Returns

Returns coordinates.

## Examples

This example creates a cell reference:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create ref lab6 inv -100.5u -90.5u 0 0.0 1.0
% $L query ref lab6 0 0 -100.5u -90.5u
% $L query ref lab6 0 0 -77.5u -108.5u
{-104500 -108500 27000 36000 inv -100500 -90500 0 0.0 1.0 {}}
```

This example creates a new layout and populates it with two other layouts as well as additional polygons:

```
# Load two layouts
set Lone [layout create poly.gds]
set Ltwo [layout create spacing.gds]

# Create an empty layout
set Lnew [layout create]

# Create topcell for new layout
$Lnew create cell TOP

# Copy patterns to new layout
$Lnew create cell ONE $Lone [$Lone topcell]
$Lnew create cell TWO $Ltwo [$Ltwo topcell]

# Place cell references in new layout
```

### **\$L create ref**

---

```
$Lnew create ref TOP ONE 0 0 0 0 1
$Lnew create ref TOP TWO 0 6000 0 0 1

# Create new layer if it doesn't exist
set newlayer 10
set predicate [$Lnew exists layer $newlayer]
if {$predicate == 0 } {
    $Lnew create layer $newlayer
}

# Create polygon on new layer
$Lnew create polygon TOP $newlayer 2500 6000 7500 6000 7500 7500 4000 7500
4000 9000 2500 9000

# Export layout
$Lnew gdsout new.gds
```

### **Related Commands**

[\\$L create cell](#)

[\\$L create polygon](#)

[\\$L create wire](#)

[\\$L create layer](#)

[\\$L create text](#)

[\\$L delete ref](#)



## \$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*]]  
[-prop *attr string* [G|U]] [-yoffset *yoff*]

### Arguments

- ***cellName***  
A required argument used to define the name of the cell in which the text object will be created.
- ***layer***  
A required argument used to define the layer number or *layer\_number.datatype\_number* designation for the layer containing the text object.
- ***x1 y1***  
A required argument used to define the coordinates of the center of the new text object. By default, coordinates are in database units (dbu).  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- ***text***  
A required argument used to define the text string to be displayed. The text string can contain newline characters, in which case the text is split into multiple text strings at each newline character, and separate text objects are created for each string.
- ***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. Requires either one *or* four arguments:
  - *presentation* - Text presentation (font, vertical, horizontal)
  - *strans* - Instructions for text transformation (reflection, absolute magnification, absolute angle)
  - *magnification* - Magnification factor (real number)
  - *angle* - Angular rotation, clockwise in degrees (real number).
- **-prop *attr string* [G|U]**  
An optional flag plus associated values defining text property data. Text properties are only supported when Preserve properties are set. 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.

G (GDS) is 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.

- `-yoffset yoff`  
Optional argument used to specify the text offset in database units. If *text* contains any newline characters, each new text object is offset in increments of *yoff*.

## Returns

None.

## Examples

This example creates and modifies text objects:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create text lab22 11 10u 10.5u "umText"
% $L query text lab22 0 0 10u 10.5u
11 10000 10500 umText
#
% $L modify text lab22 11 10u 10.5u "umText" 3 10.5u 11.5u "umText"
% $L query text lab22 0 0 10u 10.5u
""
% $L query text lab22 0 0 10.5u 11.5u
3 10500 11500 umText
```

This example writes a string, changing the angle of the text to vertical:

```
layout0 create text TOPCELL 4 0 0 "Sample Text" 0 0 50270
```

## Related Commands

[\\$L create cell](#)

[\\$L create polygon](#)

[\\$L create wire](#)

[\\$L create layer](#)

[\\$L create ref](#)

[\\$L delete text](#)

## \$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 endextn]]  
[-prop attr string [G|U]] x1 y1 x2 y2 .... x_n y_n
```

### Arguments

- *cellName*  
A required argument specifying the name of the cell in which the polygon will be created.
- *layer*  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer on which to create the polygon.
- *width*  
A required argument specifying the width of the path to create. By default, width is in database units (dbu).  
  
A coordinate length can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- -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 extensionsThe 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.

G (GDS) is 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.

- *x1 y1 x2 y2 .... x\_n y\_n*

A required argument specifying the coordinates of the vertices of the polygon. By default, coordinates are in dbu.

Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.

## Returns

None.

## Examples

This example creates a path with given coordinates:

```
layout0 create wire TOPCELL 10 250 0 0 500 600 200 300
```

This example creates and queries a path object:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create wire lab3a 11 0.25u 10u 10u 10u 20.5u 25.5u 25.5u
% $L query wire lab3a 0 0 10u 20.5u
{11 250 0 {} {} 10000 10000 10000 20500 25500 25500} 2 v 0.0 {10000 20500}
```

## Related Commands

[\\$L create cell](#)

[\\$L create polygon](#)

[\\$L create text](#)

[\\$L create layer](#)

[\\$L create ref](#)

[\\$L delete wire](#)

## \$L customLayerDrawOrder

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Changes the layer drawing order to the specified ordering. The new setting is held as a preference.

The Layer palette ordering is updated in the layout viewer GUI once sorting by draw order is set. You enable sorting by draw order by either selecting **Layer > Sort By > Draw Order** or by using the drop down menu in the upper left corner of the Layer palette.

### 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 displays the current layer number ordering.

### Returns

Returns current layer number ordering, if *layer\_list* argument is not specified.

### Example

This example changes the layer drawing order. The ordering will not be updated in the layout viewer GUI unless sorting by draw order is set.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set layout [$wb cget -_layout]
layout0
% $layout customLayerDrawOrder 2 4 1 5 6 7 8 9 10
% $layout customLayerDrawOrder
2 4 1 5 6 7 8 9 10
```

### Related Commands

None.

## \$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*

A required argument specifying the name of the cell to delete.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
$lay delete cell a9500
$wb updateDisplay
```

### Related Commands

[\\$L delete layer](#)

[\\$L delete polygons](#)

[\\$L delete text](#)

[\\$L delete polygon](#)

[\\$L delete ref](#)

[\\$L delete wire](#)

## \$L delete clip

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Deletes the specified clip.

### Usage

```
$L delete clip \  
    cellName \  
    [-id id]
```

### Arguments

- *cellName*  
The name of the cell to delete clips from.
- -id *id*  
An identifier to specify a particular clip.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0  
% set lay [$wb cget -_layout]  
layout0  
$lay clips a1220  
{1 {Clip 1} 1716 52549 16921 20622 true false}  
$lay delete clip a1220 -id 1  
$wb updateDisplay
```

### Related Commands

[\\$L clips](#)

[\\$L clipsout](#)

[\\$L create clip](#)

## \$L delete duplicate

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Deletes the specified duplicates.

### Usage

**\$L delete duplicate** ref *cell\_0...cell\_n* [-out *fileName*]

### Arguments

- ref  
A required keyword specifying to delete duplicated array and cell references (AREF or SREF).
- *cell\_0...cell\_n*  
A required keyword specifying the cells to operate on. The wildcard character '\*' is supported, and it will find zero or more cell occurrences. Using it in a search such as ca\*t would find cat, cart, and chalet.
- -out *fileName*  
An optional keyword specifying the name of the output file to contain the report. Using "stdout" sends the report to the screen.

### Returns

None.

### Example

```
% layout0 delete duplicate ref lab2 -out stdout
Duplicates logfile v2011.2_0.11
References duplicate summary:
-----
Sref duplicates found : 1
Aref duplicates found : 1
```

### Related Commands

[\\$L delete cell](#)

[\\$L delete ref](#)



## \$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.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay delete layer 4
% $wb updateDisplay
```

### Related Commands

[\\$L delete cell](#)

[\\$L delete polygons](#)

[\\$L delete text](#)

[\\$L delete polygon](#)

[\\$L delete ref](#)

[\\$L delete wire](#)

## \$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***  
A required argument specifying the name of the cell containing the polygon to delete.
- ***layer***  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer containing the polygon.
- **-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.  
  
G (GDS) is 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.
- ***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. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.

### Returns

None.

### Examples

The example deletes a polygon:

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0  
% set lay [$wb cget -_layout]  
layout0  
% $lay delete polygon a1310 29 1000 0 15000 0 15000 85000 1000 85000
```

```
% $wb updateDisplay
```

This example creates and deletes a polygon object:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create polygon lab7 11 10u 10u 10u 20.5u 20.5u 20.5u 10000 10000
% $L query polygon lab7 0 0 20.5u 20.5u
{11 10000 10000 10000 20500 20500 20500 10000 10000} 4 v 0.0 {20500 20500}
#
% $L delete polygon lab7 11 10u 10u 10u 20.5u 20.5u 20.5u 10u 10u
% $L query polygon lab7 0 0 20.5u 20.5u
""
```

## Related Commands

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete text](#)

[\\$L delete wire](#)

## \$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*  
A required argument specifying the name of the cell containing the polygons to delete.
- *layer*  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer containing the polygons to be deleted.

### Returns

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0  
% set lay [$wb cget -_layout]  
layout0  
% $lay delete polygons a1220 8  
% $wb updateDisplay
```

### Example

None.

### Related Commands

[\\$L delete cell](#)

[\\$L delete polygon](#)

[\\$L delete text](#)

[\\$L delete layer](#)

[\\$L delete ref](#)

[\\$L delete wire](#)

## \$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. By default, coordinates are in database units (dbu). Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- ***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 198 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.  
  
G (GDS) is 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.

## Returns

None.

## Examples

This example deletes a cell reference:

```
layout0 delete ref TOPCELL d3400 12850 22900 0 0 1
```

This example creates and deletes a cell reference:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create ref labB inv -100.5u -90.5u 0 0.0 1.0
% $L query ref labB 0 0 -100.5u -90.5u
% $L query ref labB 0 0 -77.5u -108.5u
{-104500 -108500 27000 36000 inv -100500 -90500 0 0.0 1.0 {}}
% $L delete ref labB inv -100.5u -90.5u 0 0.0 1.0
% $L query ref labB 0 0 -77.5u -108.5u
""
```

## Related Commands

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete text](#)

[\\$L delete wire](#)

## \$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.  
Generates an error if objects cannot be deleted.

### Usage

```
$L delete text cellName layer x1 y1 text [presentation] [strans magnification angle]  
[-prop attr string [G|U]]
```

### 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. By default, coordinates are in database units (dbu).  
Coordinates can be specified as either an integer, an integer followed by the character ‘d’ for dbu, or a floating point number (or integer) followed by the character ‘u’ for microns.
- ***text***  
The text string to be deleted.
- ***presentation strans magnification angle***  
Used only if the layout is type GDS and the text has these GDSII compatible attributes associated with it and the -preserveTextAttributes state is enabled when loading the design (or using the [layout create](#) command).
  - *presentation* — Text presentation (font, vertical, horizontal).
  - *strans* — Instructions for text transformation (reflection, absolute magnification, absolute angle).
  - *magnification* — Magnification factor (real number).
  - *angle* — Angular rotation, clockwise in degrees (real number).
- **-prop *attr string* [G|U]**  
An optional flag plus associated values defining text property data. Text properties are only supported when Preserve properties are set. 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.

G (GDS) is 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.

## Usage Notes When Using Text Attributes

- Non-existent text objects cannot be deleted. This includes attempting to delete a text object that has already been deleted. The Calibre layout viewer returns the error message “invalid text specification” in this case.
- If the text object is not deleted, check the syntax of the command used to delete the text item. You must specify the text attribute fields in the **\$L delete text** command even if the text object does not explicitly have them.
- Deleting duplicate text placements requires a **\$L delete text** statement for each placement.
- To delete multiple text items:
  - a. Get the layer information for text objects in the design using **\$L textout**.
  - b. Use the layer output from the **\$L textout** command as input for the **\$L iterator (poly | wire | text)** command to get the text attributes.
  - c. Finally, use **\$L delete text** with the combined layer and attribute information to delete the text objects.

## Returns

None

## Examples

The following example deletes the text “X1”:

```
layout0 delete text TOPCELL 12 8500 9000 0 0 "X1"
```

This example script deletes all text in at the top cell level:

```
set gdsfile "./ctext.gds"
set topcell "rcode"
set L [ layout create $gdsfile -dt_expand -preservePaths
      -preserveTextAttributes -preserveProperties ]
set layers [$L layers]
for {set i 0} {$i < [llength $layers]} {incr i} {
    # Search text label in all layers
    set lay [lindex $layers $i]
    set inst_all [$L iterator text $topcell $lay range 0 end]
    set text_num [llength $inst_all]
    if { $text_num > 0 } {
        # Fnd if layer does have some text labels
    }
}
```



```
for {set j 0} {$j < $text_num} {incr j} {
    # Delete all text labels
    set inst [lindex $inst_all $j]
    set text_str [lindex $inst 0]
    set text_x [lindex $inst 1]
    set text_y [lindex $inst 2]
    set text_strans [lindex $inst 4]
    if { [string compare $text_strans ""] == 0 } {
        $L delete text $stopcell $lay $text_x $text_y $text_str
    } else {
        $L delete text $stopcell $lay $text_x $text_y $text_str \
            [lindex $inst 3] [lindex $inst 4] [lindex $inst 5]
            [lindex $inst 6]
    }
}
}
}
$L gdsout "output.gds"
```

This example creates, modifies, and deletes text objects:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create text labT 11 10u 10.5u "umText"
% $L query text labT 0 0 10u 10.5u
11 10000 10500 umText
% $L modify text labT 11 10u 10.5u "umText" 3 10.5u 11.5u "umText"
% $L query text labT 0 0 10u 10.5u
""
% $L query text labT 0 0 10.5u 11.5u
3 10500 11500 umText

% $L delete text labT 3 10.5u 11.5u "umText"
% $L query text labT 0 0 10u 10.5u
""
```

## Related Commands

[\\$L create text](#)

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete wire](#)

## \$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 endextn]]  
[-prop attr string [G|U]] x1 y1 x2 y2 .... x_n y_n
```

### Arguments

- *cellName*  
A required argument specifying the name of the cell from which the polygon will be deleted.
- *layer*  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer containing the polygon.
- *width*  
A required argument specifying the width of the path to delete. By default, widths are in database units (dbu).  
  
Coordinate lengths can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- *-pathtype type*  
A required argument specifying 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 extensionsThe 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.

G (GDS) is 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.

- *x1 y1 x2 y2 .... x\_n y\_n*

A required argument specifying the coordinates of the vertices of the polygon. By default, coordinates are in dbu.

Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.

## Returns

None.

## Examples

This example creates a path:

```
layout0 delete wire TOPCELL 10 950
```

This example creates and deletes a path object:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L create wire lab12b 11 0.25u 10u 10u 10u 20.5u 25.5u 25.5u
% $L query wire lab12b 0 0 10u 20.5u
{11 250 0 {} {} 10000 10000 10000 20500 25500 25500} 2 v 0.0 {10000 20500}
%
% $L delete wire lab12b 11 0.25u 10u 10u 10u 20.5u 25.5u 25.5u
% $L query wire lab12b 0 0 10u 20.5u
""
```

## Related Commands

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L exists cell](#)

## \$L disconnect

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Disconnects connectivity layers previously created with the [\\$L connect](#) command. Only original layout layers can be used with this command. This command allows you to modify layer connectivity during a layout viewer session without exiting DESIGNrev and restarting the tool.

Internally, layer names are translated into layer numbers, so if layer names are used to create \$L disconnect statements, a query on \$L disconnect commands will return layer *numbers*. Layer names must be loaded before using them in a \$L disconnect statement.

---

#### Caution



The \$L connect and \$L disconnect statements ignore ordering of the *layer1* and *layer2* layers. The following commands are considered identical and will result in only one connection:

\$L connect 1 2

\$L connect 2 1

In previous releases, the two statements were considered distinct.

---

Command can be used on overlays.

### Usage

**\$L disconnect** *layer1 layer2* [by *layer3*] [via *viacell*] [width *width*]

### Arguments

- *layer1*  
The first layer to disconnect.
- *layer2*  
The second layer to connect.
- by *layer3*  
An optional third layer used to contain shapes that connect the two layers.
- via *viacell*  
An optional layer used to specify the cell name of a via cell.
- width *width*  
An optional size used to specify the width of the box dropped on the layer to connect different segments of a multi-layer path. By default, widths are in database units (dbu).

### Returns

None

## Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
%
% $lay connect 1 3
% $lay connect 2 4 by 8 width 10.
% $lay connect 10 12 via 16.
%
% $lay disconnect 1 3
% $lay disconnect 2 4 by 8 width 10.
% $lay disconnect 10 12 via 16.
```

## Related Commands

[\\$L connect](#)

## \$L duplicate cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Duplicates a cell, and adds it to the layout. The command is passed the name of a cell, which it then duplicates.

### Usage

**\$L duplicate cell** *cellname*

### Arguments

- *cellname*  
The name of the cell to duplicate.

### Returns

Name of duplicated cell.

### Examples

```
layout0 duplicate cell c1250
```

### Related Commands

[\\$L create cell](#)

[\\$L delete cell](#)

## \$L exists cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Checks to see if the specified cell exists in the layout.

### Usage

**\$L exists cell** *cellName*

### Arguments

- *cellName*  
The name of the cell to check for.

### Returns

Returns 1 if the cell exists, 0 if it does not.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay exists cell a2340
1
% $lay exists cell abc123def456
0
```

### Related Commands

[\\$L exists layer](#)

## \$L exists layer

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Checks to see if the specified layer exists in the layout.

### Usage

**\$L exists layer** *layer\_num*

### Arguments

- *layer\_num*

The layer number or *layer\_number.datatype\_number* designation for the layer to check for.

### Returns

Returns 1 if the layer exists, 0 if it does not.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay exists layer 10
1
% $lay exists layer 123456789
0
```

### Related Commands

[\\$L exists cell](#)

[\\$L isLayerEmpty](#)




## \$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 of the specified cell across all cells. If the cell contains references to other cells, those references are left intact.

The cell definition itself is left intact and is not deleted from the design.

 **Note** *Expanding* a cell specifies the cell whose instances is to be expanded one level into the space of the cells in which the expanded instances are placed. In expanding a cell, any underlying hierarchy of the cell remains intact.

*Flattening* a cell specifies the cell whose instance is to be flattened up to the level of the parent cells. The cell and its underlying hierarchy are replaced by a flattened view of the shapes.

### Usage

**\$L expand cell *cellName***

### Arguments

- *cellName*

The name of the cell to flatten.

### Returns

None.

### Examples

The following example expands a cell:

```
# Before:
#   topcell -> a -> b -> c -> d
#   topcell -> b -> c -> d
#   topcell -> c -> d

set inlayout [layout create "../input.oas" -dt_expand \
  -preservePaths -preserveTextAttributes]
$inlayout expand cell b
$inlayout oasisout out.oas

# After:
#   b -> c -> d
#   topcell -> a -> c -> d
#   topcell -> c -> d
```

And this example *flattens* a cell:

```
# Before:
#   topcell -> a -> b -> c -> d
#   topcell -> b -> c -> d
#   topcell -> c -> d

set inlayout [layout create "../input.oas" -dt_expand \
    -preservePaths -preserveTextAttributes]
$inlayout flatten cell b
$inlayout oasisout out.oas

# After:
#   d
#   topcell -> a -> b
#   topcell -> b
#   topcell -> c
```

## Related Commands

[\\$L expand ref](#)

[\\$L flatten cell](#)

## \$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.

### Returns

None.

### Example

```
% info nameofexecutable  
/clbr_latest/ic/ixl/Mgc_home/pkgs/icwb/pvt/calibrewb  
% layout0 expand ref r172 b340 230 430 0 0 1
```

### Related Commands

[\\$L expand cell](#)

[\\$L flatten ref](#)

## **\$L file**

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### **Description**

Displays the filename for the current layout.

### **Usage**

**\$L file**

### **Arguments**

None.

### **Returns**

Returns the filename for this layout. If the layout was not created by loading in data from an existing file, it returns the file handle.

If the “layout create -files” command is used to merge files during loading, an unexpected pathname will be returned with the file handle appended to the current working directory.

### **Example**

```
% layout0 file  
/home/labs/drvlab/lab2.gds
```

### **Related Commands**

None.

## \$L flatten cell

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Flattens all levels of hierarchy of the specified cell across all instances of the cell.

You would flatten the hierarchy of a cell when you want to include the cell contents directly into the design, allowing you to edit the geometry without affecting the parent cell, if not flattening the top cell. Flattening a cell cannot be undone, and it can only be done one cell at a time.

#### Note



*Flattening* a cell specifies the cell whose instance is to be flattened up to the level of the parent cells. The cell and its underlying hierarchy are replaced by a flattened view of the shapes.

*Expanding* a cell specifies the cell whose instances is to be expanded one level into the space of the cells in which the expanded instances are placed. In expanding a cell, any underlying hierarchy of the cell remains intact.

### Usage

**\$L flatten cell** *cellname* [-withDelete]

### Arguments

- *cellname*  
The name of the cell to flatten.
- -withDelete  
An optional flag which causes *newly unreferenced* cell definitions to be deleted from the design as a result of the flatten operation.

### Returns

None.

### Examples

The following example flattens a cell:

```
# Before:
#   topcell -> a -> b -> c -> d
#   topcell -> b -> c -> d
#   topcell -> c -> d

set inlayout [layout create "../input.oas" -dt_expand \
    -preservePaths -preserveTextAttributes]
$inlayout flatten cell b
$inlayout oasisout out.oas
```

```
# After:
#     d
#     topcell -> a -> b
#     topcell -> b
#     topcell -> c
```

This example expands a cell:

```
# Before:
#     topcell -> a -> b -> c -> d
#     topcell -> b -> c -> d
#     topcell -> c -> d

set inlayout [layout create "../input.oas" -dt_expand \
    -preservePaths -preserveTextAttributes]
$inlayout expand cell b
$inlayout oasisout out.oas

# After:
#     b -> c -> d
#     topcell -> a -> c -> d
#     topcell -> c -> d
```

This final example flattens the layout:

```
# Before:
#     topcell -> a -> b -> c -> d
#     topcell -> b -> c -> d
#     topcell -> c -> d

set inlayout [layout create "../input.oas" -dt_expand \
    -preservePaths -preserveTextAttributes]
$inlayout layout copy $inlayout
$inlayout oasisout out.oas

# After:
#     topcell
```

## Related Commands

[\\$L expand cell](#)

[\\$L flatten ref](#)

## \$L flatten ref

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Flattens all levels of polygons throughout the reference.

You would expand a cell reference when you want to include the cell contents directly into the design. This places the cell geometry at the top level of the hierarchy and allows you to edit the geometry without affecting the original cell.

Expanding a cell reference cannot be undone, and it can only be done one cell reference instance at a time. No other cell references are affected by an individual expansion. Once a cell reference is expanded, it is no longer associated with the cell reference information and will not be updated by future changes to that cell.

### Usage

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

### Arguments

- ***incell***  
The cell containing the reference.
- ***refcell***  
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.
- **-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.  
  
G (GDS) is 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.

### Returns

None.

## Example

The following example flattens a reference:

```
set inlayout [layout create "../input.oas" -dt_expand \  
    -preservePaths -preserveTextAttributes]  
$inlayout flatten ref topcell b 4000 2000 0 0 1  
$inlayout oasisout out.oas
```

## Related Commands

[\\$L expand ref](#)

[\\$L flatten cell](#)



## \$L gdsout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Writes the layout to a GDS-formatted file. This method does not work in read-only mode.

It prints the name of the output GDSII file to the transcript.

### Usage

```
$L gdsout fileName [.gz | .Z] [cellName] [-place] [-texttype texttype]  
[-map L [layer [datatype]]] * [-maxPolygonVertices value] [-maxPathVertices value]  
[-depth depth] [-fixPolygons] [-noEmptyCells [noRefs]] [-log logfile]  
[-clips name] [-clipsAll] [-clipsSplit] [-clipsScreen cellname llx lly urx ury]
```

### Arguments

- *fileName* [.gz | .Z]  
The pathname for the file to which the layout will be written. If you append a .gz extension to the file name, the file will be saved using gzip compression. If you append a .Z to the file name, the file will be saved using the “compress” command.
- *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.)

- **-depth** *depth*

An optional keyword and associated depth used to allow writing a layout to a specific cell hierarchy depth. Referenced cells above *depth* are written with full hierarchy. Referenced cells at *depth* are not written, but references to them are written. Cells referenced first below *depth* are not written, and their references are not written.

- **-fixPolygons**

An optional keyword that instructs the tool to fix non-orientable (self-intersecting) polygons. The vertices are rearranged so that the polygon is no longer self-intersecting, or such that the polygon is split into multiple polygons. Choosing this option results in a performance penalty, therefore the default is to not fix polygons.

- **-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.

- **-log** *logfile*

An optional keyword used to write the information to a log file. Specify the file name using *logfile*. To print the contents to the terminal window, specify “stdout” for *logfile*. The **-log** option only works if the input file is GDS.

- **-clips** *name*

An optional keyword and associated *name* of a clip that instructs the tool to write the clip to a GDS file, flattening the layout and cutting intersecting geometries.

- **-clipsAll**

An optional keyword that instructs the tool to write all clips to a GDS file, flattening the layout and cutting intersecting geometries.

- **-clipsSplit**

An optional keyword that instructs the tool to write all clips to a GDS file, generating a new file for each clip. The layout is flattened and intersecting geometries are cut. Each file written follows this rule for its name: *<design\_name>\_<clip\_name>*

- **-clipsScreen *cellname llx lly urx ury***

An optional switch that, if specified, saves a clip version for a layout. The switch can either save a screen view or a selected rectangular region. The coordinates specified are lower left followed by upper right. Coordinates are in database units (dbu).

## Returns

None

## Example

```
% basename [info nameofexecutable]
calibrewb
% layout0 gdsout out.gds
Writing GDSII file: "out.gds"
```

## Related Commands

[\\$L oasisout](#)

## \$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.

### Returns

None.

### Example

```
layout0 gridsnap 20
```

## Related Commands

None.

## \$L holdupdates

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Holds cell bounding box (bbox) calculations. Bbox calculations are always performed for operations that modify the database, however for mass edits, you might want to postpone the bbox updates until the end of a Tcl procedure.

The **\$L holdupdates** command holds the updates until the end of the Tcl procedure that the command was called in. Cascaded calls are allowed, and updates are held back until the end of the procedure of the first call to **\$L holdupdates**.

If **\$L holdupdates** is called, it automatically calls the function which resumes cell bbox calculations, **\$L allowupdates**, at the end of a Tcl procedure, therefore it is not necessary to pair **\$L holdupdates** and **\$L allowupdates** commands by calling **\$L allowupdates** directly. Use of **\$L allowupdates** is optional. Calling the **\$L allowupdates** function directly gives you a way to allow updates earlier than the end of the Tcl procedure.

---

#### Note



If the bbox updates are postponed by the **\$L holdupdates** command, queries that return the bbox of the layout could potentially be incorrect as changes which modify the bbox of cells have not yet been executed.

---

---

#### Caution



**\$L holdupdates** can *only* be called from within a Tcl procedure. It cannot be called from the command line.

---

### Usage

**\$L holdupdates**

### Arguments

None.

### Returns

None.

### Example

In this example, the `create_layout_and_add_shapes_sample1` procedure calls the `create_many_shapes` procedure, which *holds* bbox calculations. After the end of this procedure, bbox calculation is triggered, and queries to get the cell bbox are now correct.

The `create_layout_and_add_shapes_sample2` procedure suspends bbox calculations until the call to allow updates. Any query of the bbox is wrong until the trigger of the bbox calculation with `allowupdates`.

```
proc create_layout_and_add_shapes_sample1 {} {  
    set lay [layout create]  
    $lay create cell TOP  
  
    create_many_shapes $lay TOP  
  
    # Returns correct bbox  
    set bbox [$lay bbox TOP]  
}  
  
proc create_many_shapes { lay cell } {  
    $L holdupdates  
  
    $lay create polygon $cell ...  
    $lay create polygon $cell ...  
    $lay create polygon $cell ...  
}  
  
proc create_layout_and_add_shapes_sample2 {} {  
    set lay [layout create]  
    $lay create cell TOP  
  
    $L holdupdates  
  
    create_many_shapes $lay TOP  
  
    # Returns wrong bbox  
    set bbox [$lay bbox TOP]  
  
    $L allowupdates  
  
    # Returns correct bbox  
    set bbox [$lay bbox TOP]  
}
```

## Related Commands

[\\$L allowupdates](#)

## \$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 with 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*  
A required switch specifying TRUE or FALSE, which sets 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** — 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 preserve 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 preserve 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.
- **-ignoreDuplicateRefs**  
An optional flag. If specified, and the exact placements already exist, the command does not append them.

## Returns

None.

## Example

The following object method imports all cells from another layout, and combines them.

```
layout0 import layout layout2 FALSE append
```

## Related Commands

None.

## \$L instancedbout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Writes out instances of the specified cell to an ASCII results database as bounding boxes. This command writes a single bounding box for an entire AREF (array of cell references).

### Usage

**\$L instancedbout** *fileName cellName* [*inCell*]

### Arguments

- *fileName*  
The pathname for the ASCII results database file to which the cell data will be written. This file is not appendable by subsequent runs of this command.
- *cellName*  
The name of the cell to search for whose instances will be written to the ASCII results database file.
- [*inCell*]  
The name of a cell other than the top cell from which to write the instance specified by *cellName*.

### Returns

None.

### Example

```
layout0 instancedbout frs.results TYcell
```

### Related Commands

[\\$L asciiout](#)

## \$L isLayerEmpty

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Determines if no objects exist on the layer in the layout.

### Usage

**\$L isLayerEmpty** *layer\_num*

### Arguments

- *layer\_num*

The layer number or *layer\_number.datatype\_number* designation for the layer to check for.

### Returns

Returns 1 if no objects exist on the layer, and 0 for any existing objects found on *layer\_num*.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay isLayerEmpty 10
1
% $lay isLayerEmpty 60
0
```

### Related Commands

[\\$L exists layer](#)

## \$L ismodified

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Determines if there have been any modifications made to the layout since the last save was performed.

### Usage

**\$L ismodified**

### Arguments

None.

### Returns

Returns 1 if modifications made to layout since the last save, or 0 if no changes.

### Examples

```
% $lay ismodified
1
% $lay gdsout saved.gds
Writing GDSII file: "saved.gds"
% $lay ismodified
0
```

### Related Commands

None.

## \$L isOverlay

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Determines if the layout is an overlay.

### Usage

**\$L isOverlay**

### Arguments

None.

### Returns

Returns 1 if the layout is an overlay. Otherwise, returns 0.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
```

**DESIGNrev GUI > Layouts pulldown > overlay0**

```
% set O [$wb cget -_layout]
overlay0
```

**DESIGNrev GUI > Layouts pulldown > layout0**

```
% set lay [$wb cget -_layout]
layout0
%
% $O isOverlay
1
% $lay isOverlay
0
```

### Related Commands

[\\$O layoutHandle](#)

[\\$O overlayCells](#)

[layout overlays](#)

[\\$L layerFilters](#)

[\\$O layouts](#)

[\\$O overlayout](#)

[\\$L isReferenced](#)

## \$L isReferenced

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Determines if the layout is referenced in one or more overlays.

### Usage

**\$L isReferenced**

### Arguments

None.

### Returns

Returns 1 if the layout is referenced in one or more overlays. Otherwise, returns 0.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
```

**DESIGNrev GUI > Layouts pulldown > overlay0**

```
% set O [$wb cget -_layout]
overlay0
```

**DESIGNrev GUI > Layouts pulldown > layout0**

```
% set lay [$wb cget -_layout]
layout0
%
% $O isReferenced
0z=
% $lay isReferenced
1
```

### Related Commands

[\\$O layoutHandle](#)

[\\$O overlayCells](#)

[layout overlays](#)

[\\$L layerFilters](#)

[\\$O layouts](#)

[\\$O overlayout](#)

[\\$L isOverlay](#)

## \$L iterator (poly | wire | text)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays list of objects of the indicated type in the indicated cells and layers. This can be used to return all instance locations within the topcell's coordinate space.

#### Caution



Previous to Calibre release 2011.3, the **\$L iterator** command returned strings, and layout viewer Tcl scripts passed these returned strings on to Tcl commands expecting true Tcl lists. Under most conditions, script commands receiving a string treated as a Tcl list (rather than receiving a true Tcl list) would have no issues. However, occasionally these returned strings led to floating point inaccuracies, improper handling of special characters, poorly formatted lists, inconsistencies between formatted objects, and unnecessary memory used to hold the strings. To correct these issues, the **\$L iterator** command has been updated to return true Tcl lists containing Tcl\_Obj objects.

---

### Usage

**\$L iterator {poly | wire | text} cell layer range startRange endRange**  
[-depth startDepth endDepth [-filterCell cellname]]

### 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 examine.

- **layer**

The *layer\_number* or *layer\_number.datatype\_number* designation for the layer containing the objects to count.

- **range startRange endRange**

A required keyword (**range**) followed by a range of elements to return. The list returned contains all elements in the original list with indices *startRange* through *endRange*, inclusive. Indexes start at 0 for the first element. *endRange* can also be 'end' to specify the last element in the list instead of an integer.

- **-depth** *startDepth endDepth*

An optional keyword used to specify the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in cell. When traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations, provided the depth is deep enough.

- **-filterCell** *cellName*

An optional keyword used to return objects only pertaining to *cellName*. This option is only supported when **-depth** is specified.

## Returns

**Returns a Tcl list containing Tcl objects.** Floating point values are returned as Tcl floating point objects. Special characters requiring escaping are escaped as necessary. For example, a string object containing a single '}' are escaped to prevent Tcl from parsing the } as a closing string or command clause. Strings containing special Tcl characters, such as cell names containing \$ or [1], and property text strings containing blanks, are enclosed in {}'s when necessary.

This command has two sets of output formats, one where hierarchy data is returned and the other where cell contents are ignored. Hierarchy data is returned when the **-depth** argument is specified.

### Non-Hierarchical Data

- Polygon — for each polygon, returns:

`{{[properties]} {x1 y1 x2 y2 ... xn yn}}`

where:

- `{properties}` — a list of property attributes returned if the layout was read with **-preserveProperties**. If the object has no properties associated with it when using **-preserveProperties**, an empty list is returned. Without specifying **-preserveProperties**, the list value is absent from the data.
- `{x1 y1 x2 y2 ... xn yn}` — ordered list of coordinates of the polygon vertices. Coordinates are in database units (dbu).

---

### Note



One of the more common uses for this command is to create a copy of an existing layer; see the Examples section at the end of this reference page for important information.

---

- Wire — for each wire, returns:

`{width pathtype bgnextn endextn [{properties]} {x1 y1 x2 y2 ... xn yn}}`

where:

- *width* — path width, in database units (dbu).
- *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 attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned. Without specifying `-preserveProperties`, the list value is absent from the data.
- *{x1 y1 x2 y2 ... xn yn}* — ordered list of coordinates of the path vertices. Coordinates are in database units (dbu).
- 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:
  - *x y* — the coordinates of the text. Coordinates are in database units (dbu).
  - *text* — the string displayed as text.
  - *[presentation [strans magnification angle]]* — text attribute values, returned only if the text has these attributes associated with it.

#### Hierarchical Data

- Polygon — for each polygon, returns:  
`{{{properties}}} {x1 y1 x2 y2 ... xn yn}} path {bbox}}`  
 where:
  - *{properties}* — a list of property attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned. Without specifying `-preserveProperties`, the list value is absent from the data.
  - *{x1 y1 x2 y2 ... xn yn}* — ordered list of coordinates of the polygon vertices. Coordinates are in database units (dbu).
  - *path {bbox}* — the path identifying the cell in which the polygon exists and the cell's coordinate space as *{x y width height}*.
- Wire — for each wire, returns:  
`{{width pathtype bgnextn endextn [{properties}] {x1 y1 x2 y2 ... xn yn}} path {bbox}}`  
 where:
  - *width* — path width, in dbu.
  - *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 attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned. Without specifying `-preserveProperties`, the list value is absent from the data.
- *{x1 y1 x2 y2 ... xn yn}* — ordered list of coordinates of the path vertices. Coordinates are in database units (dbu).
- *path {bbox}* — the path identifying the cell in which the wire exists and the cell's coordinate space as *{x y width height}*.
- 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]]} parent {bbox}}*

where:

- *x y* — the coordinates of the text. Coordinates are in database units (dbu).
- *text* — the string displayed as text.
- *[presentation [strans magnification angle]]* — text attribute values, returned only if the text has these attributes associated with it.
- *parent {bbox}* — the path identifying the parent cell in which the text exists and the cell's coordinate space as *{x y width height}*.

## Examples

Due to the manner in which **\$L iterator poly** returns design information, there are additional steps required to retrieve polygons from a specific layer in a cell. The following suggested code is a method that will copy coordinates from layer 2 in the cell called “nand”:

```
set L [layout create lab2.gds -preserveProperties]
set clist [$L iterator poly nand 2 range 0 end]
```

In designs with `-preserveProperties` specified, and with no properties attached to the polygons, the coordinates are returned as a list starting with an empty list (denoted by curly braces) in front of the coordinates:

*{{ } x1 y1 x2 y2 .... xn yn}*

The following Tcl commands extract the coordinates:

```
$L create layer 101
foreach polygon $clist {
    eval $L create polygon topcell 101 [lrange $polygon 1 end]
}
$L gdsout out.gds
```

The Tcl **eval** command is used to convert the coordinates to separate strings. The **\$L create polygon** command requires separate strings for each coordinate.

## Related Commands

[\\$L COPY](#)

## \$L iterator (ref | sref | aref)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays a list of objects of the indicated type in the indicated cells and layers. The hierarchical traversal treats instances internal to AREFs individually. This can be used to return all instance locations within the topcell's coordinate space.

#### Caution



Previous to Calibre release 2011.3, the **\$L iterator** command returned strings, and layout viewer Tcl scripts passed these returned strings on to Tcl commands expecting true Tcl lists. Under most conditions, script commands receiving a string treated as a Tcl list (rather than receiving a true Tcl list) would have no issues. However, occasionally these returned strings lead to floating point inaccuracies, improper handling of special characters, poorly formatted lists, inconsistencies between formatted objects, and unnecessary memory used to hold the strings. To correct these issues, the **\$L iterator** command has been updated to return true Tcl lists containing Tcl\_Obj objects.

### Usage

```
$L iterator [export layer] {ref | sref | aref} cell range startRange endRange \  
[-depth startDepth endDepth [-filterCell cellname]] [-arefToSrefs] [-duplicate]
```

### Arguments

- **export *layer***  
Optionally, export your iterator result to a *layer*.
- {**ref** | **sref** | **aref**}  
A required keyword used to control which type of references are returned: single references (**sref**), arrays (**aref**), or both (**ref**).
- **cell**  
The name of the cell containing the objects to examine.
- **range *startRange endRange***  
A required keyword (**range**) followed by a range of elements to return. The list returned contains all elements in the original list with indices *startRange* through *endRange*, inclusive. Indexes start at 0 for the first element. *endRange* can also be 'end' to specify the last element in the list instead of an integer.
- **-depth *startDepth endDepth***  
An optional keyword used to specify the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in cell. When traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations, provided the depth is deep enough.

- **-filterCell** *cellName*  
An optional keyword used to return objects only pertaining to *cellName*. This option is only supported when **-depth** is specified.
- **-arefToSrefs**  
Returns SREF objects representing the internal instances of any AREFs.
- **-duplicate**  
Iterate through duplicate geometries, reporting all duplicated placements.

## Returns

**Returns a Tcl list containing Tcl objects.** Floating point values are returned as Tcl floating point objects. Special characters requiring escaping are escaped as necessary. For example, a string object containing a single ‘}’ are escaped to prevent Tcl from parsing the } as a closing string or command clause. Strings containing special Tcl characters, such as cell names containing \$ or [1], and property text strings containing blanks, are enclosed in {}’s when necessary.

This command has two sets of output formats, one where hierarchy data is returned and the other where cell contents are ignored. Hierarchy data is returned when **-depth** is specified.

### Non-Hierarchical Data

- **Sref** — Returns the cell name, coordinates, and orientation for each single reference. This command also returns any properties associated with the reference. 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 attributes returned if the layout was read with **-preserveProperties**. If the object has no properties associated with it when using **-preserveProperties**, an empty list is returned.
- **Aref** — Returns the cell name, coordinates, and orientation, plus array dimensions and spacing for each array of references. This command also returns any properties associated with the reference. 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 attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned.

### Hierarchical Data

- **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 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}]} path {bbox}}`

where:

- *cell\_name, x, y, mirror, angle, mag* — name of the referenced cell, and orientation in the design.
  - *{properties}* — a list of property attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned.
  - *path {bbox}* — the path identifying the cell in which the SREF exists and the cell's coordinate space as `{x y width height}`.
- **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 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}]} path {bbox}}`

where:

- *cell\_name, x, y, mirror, angle, mag* — name of the referenced cell, and orientation in the design. Coordinates are in database units (dbu).
- *cols rows xspace yspace* — array columns and rows, spacing along the X and Y directions.
- *{properties}* — a list of property attributes returned if the layout was read with `-preserveProperties`. If the object has no properties associated with it when using `-preserveProperties`, an empty list is returned.
- *path {bbox}* — the path identifying the cell in which the AREF exists and the cell's coordinate space as `{x y width height}`.

### Examples

This first example displays a list of objects, reporting all duplicated placements in the mask. Duplicated instances will be in the RDB file, `dup2200`, as polygons.

```
set mylayout [layout create lab2.oas]
$mylayout create layer 2200
set totcnt 0
```

```
set cells [layout peek lab2.oas -cells]
foreach cell $cells {
    set cnt [llength [$mylayout iterator export 2200 ref $cell \
        range begin end -duplicate]]
    incr totcnt $cnt
    puts "Found $cnt duplicate(s) in $cell"
}
puts "Total duplicated $totcnt"
$mylayout rdbout dup2200 e13656z -layers 2200
```

This next example illustrates floating point representation:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L1 [$cwb cget -_layout]
layout1
% $L1 iterator sref [$L1 topcell] range 0 end
{{inv -104000 -58000 0 0.0 1.0 {}} {nand 0 -22000 0 0.0 1.0 {}}}
```

This final example illustrates text string representation:

```
% set L2 [$cwb cget -_layout]
layout2
% $L2 iterator sref [$L2 topcell] range 0 end
{{inv[1]} -104000 -58000 0 0.0 1.0 {}} {{nand$1} 0 -22000 0 0.0 1.0 {}}}
```

## Related Commands

None.

## \$L iterator count (poly | wire | text)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays the number of objects of the indicated type in the indicated cells and layers.

### Usage

**\$L iterator count {poly | wire | text} *cell layer* [-depth *startDepth endDepth*  
[-filterCell *cellname*]]**

### 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.
- **-depth *startDepth endDepth***  
An optional keyword used to specify the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in cell. When traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations, provided the depth is deep enough.
- **-filterCell *cellName***  
An optional keyword used to return the count of objects pertaining only to *cellName*. This option is only supported when -depth is specified.

### Returns

Returns the number of objects of the indicated type in the indicated cells and layers.

### Example

```
% layout0 iterator count wire r2200  
2
```

### Related Commands

None.



## \$L iterator count (ref | sref | aref)

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays the number of objects of the indicated type in the indicated cells and layers. The hierarchical traversal treats instances internal to AREFs individually.

### Usage

**\$L iterator count {ref | sref | aref} cell** [-depth *startDepth endDepth*] [-filterCell *cellname*] [-arefToSrefs] [-duplicate]

### Arguments

- **{ref | sref | aref}**  
A required keyword used to control which type of references are returned: single references (**sref**), arrays (**aref**), or both (**ref**).
- **cell**  
The name of the cell containing the references to count.
- **-depth *startDepth endDepth***  
An optional keyword used to specify the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in cell. When traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations, provided the depth is deep enough.
- **-filterCell *cellName***  
An optional keyword used to return the count of objects only pertaining to *cellName*. This option is only supported when -depth is specified.
- **-arefToSrefs**  
Returns SREF objects representing the internal instances of any AREFs.
- **-duplicate**  
Iterate through duplicate geometries, reporting all duplicated placements.

### Returns

Returns the number of objects of the indicated type in the indicated cells and layers.

### Example

None.

### Related Commands

None.

## \$L layerconfigure

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Configures a layer with the specified layer properties.

### Usage

**\$L layerconfigure** [-regular | -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 -regular.

- **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 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 pixels.
- -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

## Returns

List of layer properties.

## Example

```
% layout0 layerconfigure 4  
-fill red -outline red -stipple gray12 -visible 1 -linewidth 1
```

## Related Commands

None.

## \$L layerFilters

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Allows the definition of layer filters used to filter the layers shown in the Layers Browser.

### Usage

```
$L layerFilters [-mdpChipView | -mdpLevelView] -names [layer] | -layers  
filterName_0...filterName_n | -add filterName layer_0...layer_n
```

### Arguments

- -mdpLevelView | -mdpChipView  
For use with MDP jobdecks. Jobdecks support two sets of layer filters. One for each of the level layer view, and the chip layer view.
- -names [*layer*]  
Retrieve filter names defined in the layout. Optionally retrieve the filters for a specific layer within the layout.
- -layers *filterName\_0...filterName\_n*  
Retrieve a unique set of layers for the specified filter(s). Specifying multiple filters performs an OR of the filter layers.
- -add *filterName layer\_0...layer\_n*  
Add layer filter to the list of filters. *filterName* is the layout designator for the DESIGNrev layer, and *layer\_0...layer\_n* is the list of layers, separated by spaces.

### Returns

There are returns when the command is used as a query. The “\$L layerFilters -names” command returns filter names defined in the layout. The “\$L layerFilters -layers” command returns a unique set of layers for the specified filters.

### Example

```
% cat mylayout.gds.layerprops
3 red diagonal_2 altmix 1 1
4 pink speckle 4 1 1
5 orange speckle Orange 1 1
7 purple speckle 7 1 1
11 blue speckle Blue 1 1
layerFilter ma 4
layerFilter mb 4 7

% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout1
% $L layerFilters -names
# mb ma
```

```
$L layerFilters -layers mb  
# 4 7  
$L layerFilters -names 4  
# mb ma  
$L layerFilters -names 1  
# ""
```

## Related Commands

[\\$O layoutHandle](#)

[\\$O overlayCells](#)

[layout overlays](#)

[\\$L isReferenced](#)

[\\$O layouts](#)

[\\$O overlayout](#)

[\\$L isOverlay](#)

## \$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.

---

### Caution



Previous to Calibre release 2011.4, the **\$L layernames** command returned strings, and layout viewer Tcl scripts passed these returned strings on to Tcl commands expecting true Tcl lists. Under most conditions, script commands receiving a string treated as a Tcl list (rather than receiving a true Tcl list) would have no issues. However, occasionally these returned strings led to floating point inaccuracies, improper handling of special characters, poorly formatted lists, inconsistencies between formatted objects, and unnecessary memory used to hold the strings. To correct these issues, the **\$L layernames** command has been updated to return true Tcl lists containing Tcl\_Obj objects.

---

### 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} ...}
```

## Returns

Returns a list of layer number, layer name pairs.

## Example

```
% $lay layers
1 2 5
% $lay layernames 1 POLY
% $lay layernames 1
POLY
% $lay layernames
1 POLY 2 2 5 5
% $lay layernames 2 \ $M1
% $lay layernames 2
$M1
% $lay layernames
1 POLY 2 { $M1 } 5 5
```

## Related Commands

None.

## \$L layers

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays list of all layer numbers in this layout.

Layer numbers are either integers, *or* integers followed by a period and a datatype. The syntax is *layernum.datatype*. Datatypes are defined as sub-layers of the main layer number. The layout viewer drops zero datatypes, therefore 10 and 10.0 represent the same layer/datatype pair.

### Usage

**\$L layers** [-shadow | -regular | -all] [-cell *cellName*] [-name *layerName*]

### 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.

- -cell *cellName*

An optional keyword indicating the output will only be the layers used in the specified cell, *cellName*.

- -name *layerName*

An optional keyword to specify a layer name and returning the layer number (or layer number *and* datatype if one is included).

### Returns

List of all the layers in the layout, or if the -name switch is used, returns the defined layers for the specified layer name.

### Example

List the layer numbers, including sub-layers:

```
% layout0 layers
1 2.1 2.2 2.4 2.6 4 5 6 7 8 9 10 28 29 30 60
```

Return defined layers for a layer name:

```
set lay [layout create]
$lay create layer 123.4
$lay layernames 123.4 dete
```



```
$lay layers -name dmm  
120.41
```

### Related Commands

None.

## \$L layoutType

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Checks or sets the supplied layout's type property to the supplied type, useful in cases such as when a file is loaded as GDS, but saved as OASIS. No type checking is performed by this command. You use this command to indicate to the GUI **File > Save** option which layout type to save the layout as.

### Usage

**\$L layoutType** [set {gds | oasis | ascii}]

### Arguments

- set {gds | oasis | ascii}

The new type for the layout.

---

#### Note



**\$L layoutType set** is often used when creating a new layout handle.

---

### Returns

If called with no options, it returns one of the following: gds, oasis, ascii, overlay, mdp.

### Example

```
layout0 layoutType set gds
```

### Related Commands

None.

## \$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.

### Returns

Returns the GDS library name if no library name is specified, otherwise there are no returns.

### Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay libname
mentor.db
% $lay libname dawas.db
% $lay libname
dawas.db
```

### Related Commands

None.

## \$L maxdepth

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Returns the maximum depth of the selected cell in the layout.

If a cell name is not specified, the layout viewer returns the highest depth in the design.

### Usage

**\$L maxdepth** [*cellName*]

### Arguments

- *cellName*

An optional cell name for which to determine the maximum depth. The default is to return the highest depth in the design.

### Returns

The result is an integer representing the depth.

### Example

```
% layout0 maxdepth c32d
4
```

### Related Commands

None.

## \$L MASK\_LAYER\_INFO

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Calculates the area and extent of a layer. The command flattens all geometries and references of the specified cell and layers (or the layer subset, if the *endDepth* argument is supplied) to a temporary data structure, and then merges found geometries so that nothing is overlapping. The command then calculates the area and extent, based on the current database units.

This command is read-only and does not modify layers. It returns an error if a layer does not contain objects.

### Usage

**\$L MASK\_LAYER\_INFO** *cellName layer* [*endDepth*]

### Arguments

- ***cellName***  
Is a required name of a cell to start the hierarchical scan. If the specified cell does not exist, the command generates an error.
- ***layer***  
Is a required layer number to scan. If the layer contains no objects the area returned will be zero. If the layer does not exist, the command generates an error.
- ***endDepth***  
Is an optional depth down the hierarchy from the current level to complete the scan at. The default is all levels.

### Returns

The result is a list in the following format:

```
area llx lly width height
```

### Example

```
% layout0 MASK_LAYER_INFO lab2 1
46259625000.0 20250 19750 415000 368250
```

### Related Commands

None.

## \$L modify origin

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Changes the origin of a specified cell from (0, 0) to *x y*, in database units.

### Usage

**\$L modify origin** *cellName* *x y* [-anchor\_refs]

### Arguments

- *cellName*  
A required argument specifying the name of the cell whose origin is to be modified.
- *x y*  
A required argument specifying the coordinates at which to place the origin of the cell. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- -anchor\_refs  
An optional keyword and argument used to force references to stay in place.

### Returns

None.

### Examples

This example changes the origin of a cell:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay modify origin lab2 3000 2200
```

This example returns the bounding box of a cell and modifies the origin of the cell:

```
% $L bbox inv
-4000 -18000 23000 18000
% $L modify origin inv -100.0u 100.0u
% $L bbox inv
96000 -118000 123000 -82000
% $L modify origin inv 100.0u -100.0u
-4000 -18000 23000 18000
```

### Related Commands

[\\$L cells](#)

## \$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***  
A required argument specifying the name of the cell in which the original text object exists.
- ***layer***  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer containing the original text object.
- ***x1 y1***  
A required argument specifying the coordinates of the center of the original text object. By default, coordinates are in database units (dbu).  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- ***text***  
A required argument specifying the text string displayed by the original text object.
- ***cellNameNew***  
A required argument specifying the name of the cell in which the new text object will be created. (Not available in Calibre DESIGNrev).
- ***layerNew***  
A required argument specifying the layer number or *layer\_number.datatype\_number* designation for the layer containing the new text object.
- ***x1New y1New***  
A required argument specifying the coordinates of the center of the new text object. By default, coordinates are in dbu.  
Coordinates can be specified as either an integer, an integer followed by the character 'd' for dbu, or a floating point number (or integer) followed by the character 'u' for microns.
- ***textNew***  
A required argument specifying the text string to be displayed by the new text object.

### Returns

None.

### Example

None.

### Related Commands

None.



## \$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 cannot be used as 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. The output layer specified should not also be an input layer.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay NOT 2 4 101
% $wb updateDisplay
```

### Related Commands

[\\$L AND](#)

[\\$L OR](#)

[\\$L XOR](#)

## \$L oasisout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Writes the layout to a OASIS-formatted file. This method does not work in read-only mode.

### Usage

```
$L oasisout fileName [.z | .gz | .Z] [cellName] [-place] [-map L [layer [datatype]]]  
[-maxPolygonVertices value] [-maxPathVertices value] [-depth depth]  
[-fixPolygons] [-noEmptyCells [noRefs]] [-cBlocks] [-strictMode]  
[-clips name] [-clipsAll] [-clipsSplit] [-clipsScreen cellname llx lly urx ury]  
[-noRefRepetition] [-noPathRepetition] [-noPolyRepetition] [-noStringRepetition]  
[-noRepetition]
```

### Arguments

- ***fileName*** [.z | .gz | .Z]  
The pathname for the file to which the layout will be written. If you append a .z or .gz extension to the file name, the file will be saved using gzip compression. If you append a .Z to the file name, the file will be saved using the “compress” command. gzip and compress must be in your path for this compression to function.
- ***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. Layout ranges cannot be remapped.
- **-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. *The 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. *The 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.)

- **-depth** *depth*

An optional keyword and associated depth used to allow writing a layout to a specific cell hierarchy depth. Referenced cells above *depth* are written with full hierarchy. Referenced cells at *depth* are not written, but references to them are written. Cells referenced first below *depth* are not written, and their references are not written.

- **-fixPolygons**

An optional keyword that instructs the tool to fix non-orientable (self-intersecting) polygons. The vertices are rearranged so that the polygon is no longer self-intersecting, or such that the polygon is split into multiple polygons. Choosing this option results in a performance penalty, therefore the default is to not fix polygons.

- **-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.)

- **-strictMode**

Writes OASIS files in strict mode (file type OASIS-STRICT-MODE), which may improve subsequent Calibre batch run performance.

---

**Note**

Layout viewer strict mode flags are set consistently with Calibre strict mode flags.

---

- **-clips** *name*

An optional keyword and associated *name* of a clip that instructs the tool to write the clip to an OASIS file, flattening the layout and cutting intersecting geometries.

- **-clipsAll**

An optional keyword that instructs the tool to write all clips to an OASIS file, flattening the layout and cutting intersecting geometries.

- **-clipsSplit**  
An optional keyword that instructs the tool to write all clips to an OASIS file, generating a new file for each clip. The layout is flattened and intersecting geometries are cut. Each file written follows this rule for its name: *<design\_name>\_<clip\_name>*
- **-clipsScreen** *cellname llx lly urx ury*  
An optional switch that, if specified, saves a clip version for a layout. The switch can either save a screen view or a selected rectangular region. The coordinates specified are lower left followed by upper right. Coordinates are in database units (dbu).
- **-noRefRepetition**  
An optional switch that instructs the tool to prevent repeated cell placements from being collapsed into repetitions. Arrays with rotated & non-integer scaled cells are created regardless of whether -noRefRepetition is used.
- **-noPolyRepetition**  
An optional switch that instructs the tool to disable polygon compression.
- **-noPathRepetition**  
An optional switch that instructs the tool to disable path compression.
- **-noStringRepetition**  
An optional switch that instructs the tool to disable string compression.
- **-noRepetition**  
An optional switch that instructs the tool to disable reference, polygon, path, and string repetitions.

## Returns

It prints the name of the output OASIS file to the transcript.

## Example

```
% basename [info nameofexecutable]
calibrewb
% layout0 oasisout zout.oas
Writing file: "zout.oas"
```

## Related Commands

[\\$L gdsout](#)

## \$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](#)).

### Usage

**\$L options** [*report\_mode*]

### Arguments

- *report\_mode*

An optional argument specifying the option to report on. At most one of the following option arguments are 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_layer_n}
```

- **-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.

### Returns

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.

### Example

```
% $lay options  
-dtExpand
```

### Related Commands

None.

## \$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.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay OR 2 4 102
% $wb updateDisplay
```

### Related Commands

[\\$L AND](#)

[\\$L NOT](#)

[\\$L XOR](#)

## \$L pushmarkers

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview

### Description

Shifts error markers in a specified Calibre RVE ASCII database in the hierarchy to match coordinates with polygons on the relevant cell level, and outputs the modified database to a new file.

Normally, Calibre tools generate all error markers in the coordinates of the topcell and at the highest depth. This command scans the design on layers specified in the `-map` arguments and if there are shapes on that layer that interact (are touching or are overlapped by) an error marker, the marker is pushed down to that level. If multiple polygons on different levels interact with the marker, the markers are moved to the highest level that has interacting shapes.

Loading the Calibre RVE ASCII database that is output from this command will show the error markers at their new levels.

### Usage

```
$L pushmarkers input_file [-out output_file] [-cell cellName] [-only]  
    [-map {rulename1 | *} layer1 [.datatype1]]  
    [-map {rulename2 | *} layer2 [.datatype2]]  
    ...
```

### Arguments

- ***input\_file***  
A required keyword specifying the RVE ASCII database to scan.
- **-out *output\_file***  
An optional keyword specifying the output file to write. If this argument is not specified, the name *output.rdb* is used by default.
- **-cell *cellName***  
An optional keyword specifying the name of the topcell in the input RVE ASCII database. This is only used if the database topcell does not match the topcell in the layout handle.
- **-only**  
An optional switch. If specified, this command only outputs markers from rules that are named in the `-map` argument.



- `-map {rulename | *} layer[.datatype]`

An optional keyword specifying a layer (with optional datatype) to search for polygon shapes matching error markers.

- If an SVRF *rulename* is specified, it is used as a filter; only markers found by the specified *rulename* are checked on the specified layer.
- If the wildcard character ‘\*’ is specified instead of a *rulename*, all rule result markers are searched on the specified layer.

To include more than one layer in the search, each layer must have its own `-map` statement.

If no `-map` statements are specified, all markers that were on lower levels of the hierarchy are pulled to the topcell level in the output.

## Returns

None.

## Example

```
layout0 pushmarkers lab1.drc.results
```

## Related Commands

None.

## \$L query

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Selects the requested element nearest to the given (X,Y) coordinate and displays 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 in database units.

### Caution



Previous to Calibre release 2011.2, the **\$L query** command returned strings, and layout viewer Tcl scripts passed these returned strings on to Tcl commands expecting true Tcl lists. Under most conditions, script commands receiving a string treated as a Tcl list (rather than receiving a true Tcl list) would have no issues. However, occasionally these returned strings led to floating point inaccuracies, improper handling of special characters, poorly formatted lists, inconsistencies between formatted objects, and unnecessary memory used to hold the strings. To correct these issues, the **\$L query** command has been updated to return true Tcl lists containing Tcl\_Obj objects.

---

### Usage

**\$L query** {**ref** | **polygon** | **wire** | **edge** | **vertex**| **closept** | **text**} *cell startDepth endDepth x y*  
[*maxSearchDistanceX maxSearchDistanceY*] [-path] [-list]

### 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. By default, coordinates are in database units (dbu).  
  
Coordinates can be specified as either an integer, an integer followed by the character ‘d’ for dbu, or a floating point number (or integer) followed by the character ‘u’ for microns.
- **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.  
  
This argument acts as a filter on the results; if no objects meet the filter criteria (are located within the specified distance), no results are returned.
- **-path**  
An optional keyword used to append the hierarchical path to the returned data. This option only works for GDS and OASIS layouts, and it does not apply to MDP or view-only OASIS data.
- **-list**  
An optional keyword which causes the return list to always contain sublists of the matching database objects. This assists with the processing of multiple returned objects. The **query ref**, **query edge**, and **query vertex** return object types already behave in this manner and are unaffected.

## Returns

**Returns a Tcl list containing Tcl objects.** Floating point values are returned as Tcl floating point objects. Special characters requiring escaping are escaped as necessary. For example, a string object containing a single ‘}’ are escaped to prevent Tcl from parsing the } as a closing string or command clause. Strings containing special Tcl characters, such as cell names

containing \$ or [1], and property text strings containing blanks, are enclosed in {}'s when necessary.

Returns are specific to the object type being returned:

- **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. For returned X and Y values, the coordinate space of the specified cell is used.
  - *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.

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 second x in the edges x,y pair.
- **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.

## Examples

These first three examples illustrate floating point representation:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout0

% $L query ref [$L topcell] 0 0 0 0
{-40000 -102000 70000 122000 nand 0 -22000 0 0.0 1.0 {}}

% $L query text [$L topcell] 0 0 25000 25000
3005 10000 10000 GROUND 8 0 2.0 0.0
```

These examples illustrate text string representation:

```
% $L query text [$L topcell] 0 0 -85 -85
3006 -85 -87 {text string with blanks}

% $L query text [$L topcell] 0 0 10000 10000
3005 10000 10000 VCC

% $L query ref [$L topcell] 0 0 -104000 -58000
{-108000 -76000 27000 36000 {inv[1]} -104000 -58000 0 0.0 1.0 {}}

% $L query polygon [$L topcell] 0 0 -125000 -13000
```

```
{1 {{1 TestProp1}} -134000 -15000 -110000 -150000 -110000 -11000 -134000  
-11000} 6 e 2000.0 {-125000 -11000}
```

The final example illustrates escaped characters:

```
% $L query ref [$L topcell] 0 0 -104000 -58000  
{-108000 -76000 27000 36000 \}_inv -104000 -58000 0 0.0 1.0 {}}
```

## Related Commands

None.

## \$L rdbout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Convert all polygons on a layer into RDB output.

### Usage

```
$L rdbout rdb_filename -cell cellName -layers layer_0...layer_n [-hier] [-properties]
```

### Arguments

- ***rdb\_filename***  
A required keyword specifying the output RVE ASCII database to create.
- **-cell *cellName***  
A required keyword specifying the name of the cell to start scanning the layer from.
- **-layers *layer\_0...layer\_n***  
A required keyword specifying the layers to scan.
- **-hier**  
An optional switch specifying to scan in hierarchical mode. The default is flat.
- **-properties**  
An optional switch to specify preserving properties when scanning layers.

### Returns

None.

### Example

```
layout0 rdbout mylayer1.rdb -cell lab2 -layers 1
```

### Related Commands

[\\$L report](#)



## \$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. This is the default.
- **1**  
Layout is read-only.

### Returns

Returns a zero or one when used as a query, otherwise there are no returns.

### Examples

```
% layout0 readonly
0
% layout0 readonly 1
% layout0 readonly
1
```

### Related Commands

None.

## \$L refcount

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Retrieve the number of references to a specified cell or a total count of the instances beneath a specified cell.

### Usage

**\$L refcount** [-parent *cell*] [*cellName*]

### Arguments

- -parent *cell*  
An optional keyword specifying to return the number of instance in *cell*.
- *cellName*  
An optional keyword specifying to return the number of instances of *cellName* in the design or in a specified parent cell.

### -Returns

None.

### Examples

This example returns the number of instances in *cellb40*.

```
$L refcount -parent cellb40
```

This example returns the number of instances of *celli01* in the design.

```
$L refcount celli01
```

This example returns the number of instances of *cellchild* in *topcell*.

```
$L refcount -parent topcell cellchild
```

### Related Commands

[layout peek](#)

## \$L report

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Generate report information about a layout in ASCII format.

### Usage

**\$L report duplicate ref *cell\_0...cell\_n* -out *fileName***

### Arguments

- **duplicate ref *cell\_0...cell\_n***  
A required keyword specifying to report duplicated placements of the specified cell(s). The wildcard character '\*' is supported, and it will find zero or more cell occurrences. Using it in a search such as ca\*t would match cat, cart, and caught.
- **-out *fileName***  
A required keyword specifying the name of the file to contain the ASCII report. Using "stdout" sends the report to the screen.

### -Returns

None.

### Example

```
% layout0 report duplicate ref * -out stdout
Duplicates logfile v2011
References duplicate summary:
-----
Sref duplicates found : 1
Aref duplicates found : 1
```

### Related Commands

[\\$L rdbout](#)

## \$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.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout4
$lay scale 2
% $wb updateDisplay
```

### Related Commands

None.

## \$L SIZE

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Sizes the specified layers and writes the output to the layer **Lout**.

#### Caution



The SIZE operation is performed on a cell-by-cell basis, and input shapes are merged on a per cell basis. This command cannot be used as a hierarchical operation.

### Usage

**\$L SIZE *Lin Lout BY val***

#### Note



SIZE and BY must be capitalized.

### Arguments

- ***Lin***

The layer number of the layer to be sized.

- ***Lout***

The layer number of the output layer for the operation.

#### Note



If you are sizing a single layer, you can set ***Lin*** and ***Lout*** to the same layer, however the original unsized geometries will remain.

- ***BY val***

A required keyword/argument pair used to specify the amount, in microns, by which the layer is sized. Sizing is performed on a cell-by-cell basis. Hierarchical interactions are not accounted for.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
%
% set lay [$wb cget -_layout]
layout0
%
% $lay SIZE 1.1 2.1 BY 0.01
```

Sized layer by 0.01

### Related Commands

None.

## \$L srefsFromAref

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Takes the formatted output from the **\$L query ref** command when run on an AREF, displaying the list of SREF data formatted in the output format from the **\$L query ref** command for an SREF.

Coordinates and coordinate lengths can be specified as either an integer, an integer followed by the character 'd' for database units (dbu), or a floating point number (or integer) followed by the character 'u' for microns.

If you are taking the output directly from the **\$L query ref** command, you may need remove the enclosing curly braces before entering it as an argument to this command.

### Usage

**\$L srefsFromAref** *llx lly xlen ylen cellname x y mirror angle mag cols rows dx dy*  
[*{properties}*] [-noBbox]

### Arguments

- **llx lly** — A required argument which specifies the lower left corner of the bounding box of the reference. By default, coordinates are in database units (dbu).
- **xlen ylen** — A required argument which specifies the width and height of the bounding box of the reference. By default, coordinate lengths are in dbu.
- **cell\_name x y mirror angle mag** — A required argument which specifies the name of the referenced cell, coordinates, and orientation in the design. By default, coordinates are in dbu.
- **cols rows dx dy** — A required argument which specifies the array information: dimensions (cols rows) and spacing (dx dy). By default, coordinates are in dbu.
- **{properties}** — An optional list of property attribute name / value pairs, returned only if the layout is type GDS and the ref has properties associated with it. Ignored for the current release if specified.
- **-noBbox** — An optional argument specifying output does not include the transformed SREF bbox data.

### Returns

A list of SREFs. Each SREF is enclosed in braces ({}).

### Examples

This example returns a list of SREFs:

```
set file "mylayout.oas"  
set topcell "TOPCELL"  
set W [find objects -class cwbWorkBench]  
set lay [layout create $file]
```

```
# last two arguments are coordinates
set qresult [$lay query ref $topcell 0 0 95840 12280]
set aref [split [lindex $qresult 0]]

proc getSrefsFromAref { lay aref } {
    eval $lay srefsFromAref $aref
}
getSrefsFromAref $lay $aref
```

This example returns a list of SREFs converted from an AREF string:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
% $L query ref lab2b 0 0 -77u -158u
{-10600 -19600 5700 7600 inv -10200 -17800 0 0.0 1.0 2 2 3000 4000 {}}
% foreach sref [lsort [ \
    $L srefsFromAref -106u -196u 57u 76u inv -102u -178u \
    0 0.0 1.0 2 2 30u 40u]] {
    puts "$sref"
}
-10600 -15600 2700 3600 inv -10200 -13800 0 0.0 1.0
-10600 -19600 2700 3600 inv -10200 -17800 0 0.0 1.0
-7600 -15600 2700 3600 inv -7200 -13800 0 0.0 1.0
-7600 -19600 2700 3600 inv -7200 -17800 0 0.0 1.0
```

## Related Commands

None.



## \$L textout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays all text elements in the given cell as a string.

### Usage

**\$L textout** *cellName* [-split]

### Arguments

- *cellName*  
The name of the cell containing the text to output.
- -split  
Formats the output string without curly braces surrounding the text part:  
`{{layer x y text} ... {layer x y text}}`

### Returns

Returns all text elements in the given cell as a string, formatted as follows:

`{{layer x y {text}} ... {layer x y {text}}}`

#### Note



Early (pre-2007.3) versions of this command returned text without the enclosing braces signifying unevaluated strings (`{ }`), which can cause issues with Tcl scripts expecting blank characters in the output. For backwards compatibility, use the optional -split argument.

### Example

Given a layout with the handle ar1 with the topcell TOP, the following code takes the text out of the topcell and formats the data into individual lines:

```
foreach x [ar1 textout TOP] {puts $x}
```

The output might look similar to the following output:

```
3 45500 150500 {dense_line_end...}  
3 52500 150500 {corners...}  
3 59500 150500 {corners...}  
...  
3 87500 157500 {broken_h...}
```

### Related Commands

None.

## **\$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.

### **Usage**

**\$L topcell** [all]

### **Arguments**

- all

An optional argument used to return a list of the names of all cells without references to them.

### **Returns**

Returns the name of the topcell in \$L. If \$L contains multiple topcells, returns the one with the most total descendants.

When the all option is specified, this object returns all topcells.

### **Example**

```
% layout0 topcell  
lab11
```

### **Related Commands**

None.

## \$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.

### Returns

If no arguments are used, the command returns the entire transcript.

### Example

```
layout0 transcript 1
```

### Related Commands

[\\$L transcript output](#)

[\\$L transcript range](#)

[\\$L transcript state](#)

## \$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.

### Returns

None.

### Example

```
layout0 transcript 1
layout0 transcript output trans.out
```

### Related Commands

[\\$L transcript](#)

[\\$L transcript range](#)

[\\$L transcript state](#)

## \$L transcript range

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays 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.

### Returns

A range of transcript lines.

### Example

None.

### Related Commands

[\\$L transcript](#)

[\\$L transcript output](#)

[\\$L transcript state](#)

## \$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**

### Arguments

None.

### Returns

- 1 — Transcript is being recorded.
- 0 — Transcript is not being recorded.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay transcript 1
% $lay transcript output trans.out
% $lay transcript state
1
```

### Related Commands

[\\$L transcript](#)

[\\$L transcript output](#)

[\\$L transcript range](#)

## \$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 displays simply returns the size of the database unit.

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 0.001, 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 changed if you issue the [\\$L units microns](#) command.

### Returns

When no argument is specified, returns the size of the database unit.

### Example

```
% layout0 units database  
1e-09
```

### Related Commands

[\\$L units microns](#)

[\\$L units user](#)

## \$L units microns

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Usage

**\$L units microns** [*units\_per\_micron*]

### Description

Simplifies setting **\$L units database** and **\$L units user** when the user unit size is 1 micron.



#### 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** ( $10^{-6}/\text{units\_per\_micron}$ )”:

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  $1/\text{units\_per\_micron}$ .

It does this by issuing the command “**\$L units user** ( $1/\text{units\_per\_micron}$ )”. Combined with “**\$L units database** ( $10^{-6}/\text{units\_per\_micron}$ )”, this results in a user unit size of 1 micron.

### Arguments

- *units\_per\_micron*

The number of database units per micron. There is no default value for *units\_per\_micron*.

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.

### Returns

When no argument is specified, returns current number of database **units per microns**.



## Examples

A) Issuing the command with an OASIS database:

```
$lay units microns 5
```

- Issues the command **\$L units database** ( $10^{-6}/5$ )  
or **\$L units database** 0.0000002.

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** = **0.0000002**

B) Issuing the command with a GDS database:

```
$lay units microns 5
```

- Issues the command **\$L units user** 0.2.  
This translates into 0.2 user units in a database unit, or 1 user unit equals 5 database units. Since the database unit is 0.2 microns, the user unit is 1 micron.
- Issues the command **\$L units database** ( $10^{-6}/5$ ) or **\$L units database** 0.0000002.  
This translates into .20 micron because  $10^{-6}$  is one micron, expressed in meters.

## Related Commands

**\$L units database**

**\$L units user**

## \$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.

---

 **Note** For OASIS files this command is not recommended. Use the [\\$L units microns](#) command because in OASIS 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.  
When you create a new database using batch commands, it is defined such that the ratio of user units per database unit is 0.001. 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

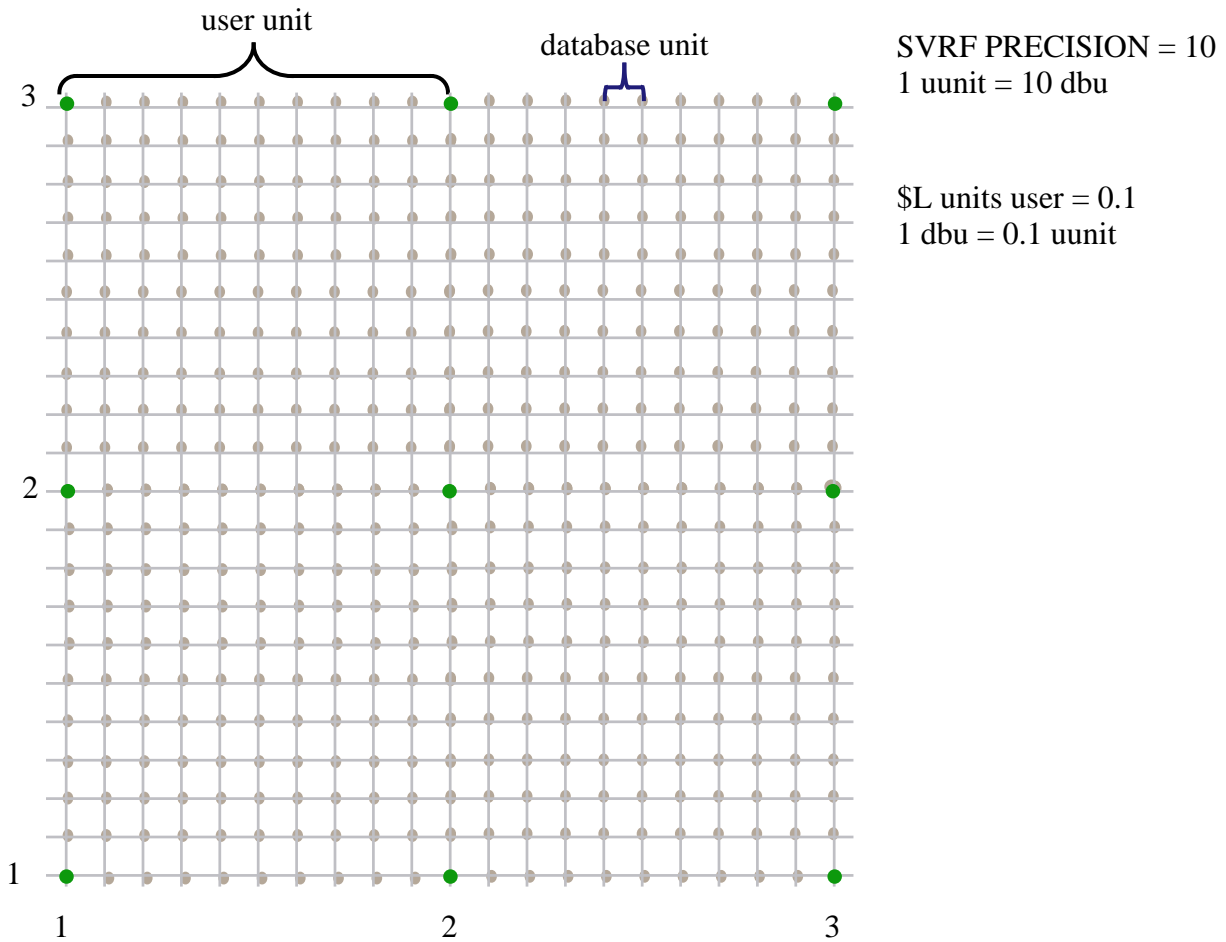
The [Precision](#) statement in SVRF defines the database precision in terms of database units per user unit. Another way to say this is that it defines the size of a database unit, expressed in user units. The default is 1000. The [\\$L units user](#) batch command also defines the database precision, but in terms of user units per database unit. Notice 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.

This is as shown in [Figure 6-3](#).

**Figure 6-3. Relating User and Database Units to the Design Grid**



## Returns

When no argument, returns the ratio of user units per database unit.

## Example

None.

## Related Commands

[\\$L units database](#)

[\\$L units microns](#)

## **\$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**

### **Arguments**

None.

### **Returns**

None.

### **Example**

```
layout0 update
```

### **Related Commands**

None.

## \$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.

---

#### Note



This is an edit command, and it must be run at the layout level rather than from an overlay.

---

### Usage

**\$L viacell** *cellname*

### Arguments

- *cellname*

The name of the cell to be set as a via cell.

---

#### Note



Specifying an asterisk (\*) in the cell name will match zero or more cell occurrences, and using it in a search such as ca\*t would find cat, cart, and cataract.

---

### Returns

None.

### Example

```
$lay viacell c4400
```

### Related Commands

None.

## \$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.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay XOR 2 4 103
% $wb updateDisplay
```

### Related Commands

[\\$L AND](#)

[\\$L NOT](#)

[\\$L OR](#)

## Peek Object

Peeking refers to probing for layout information without loading a layout file into memory. Peek objects are created with the [layout peek](#) command which displays units, precision, list of cells, list of topcells, list of layers, and other information about layout files. The [layout peek](#) command supports two options that allow *peek handles* to be generated:

- **-handle** is used to indicate the handle to assign to the newly created peek handle. Multiple [layout peek](#) queries can use this option to avoid rereading layout files to extract the peek information.
- **-cache** is used to specify the location of a cache file to prevent rereading the layout to retrieve this information.

## \$P delete

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Deletes the peek handle defined in a previous [layout peek](#) -handle command, to release memory.

### Usage

**\$P delete** [-preservercache]

### Arguments

- **-preservercache**  
An optional flag that prevents the cache file from being removed upon deletion of the peek handle.

### Returns

None

### Example

This example create a peek handle, uses it, and then deletes it:

```
set layout [lindex lab2.gds]

# Create peek handle and use cache.
set peek [layout peek $layout -handle mypeek -cache . ]

# Read out data and dump to shell.
array set peekdata [$peek peek -type -cellcount]
puts "Cell count: $peekdata(cellcount) "

# Delete the peek handle, preserving cache for subsequent runs.
$peek delete -preservercache
```

## Related Commands

[layout peek](#)

[\\$P file](#)

[\\$P peek](#)



## \$P file

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Returns the file name for a peek handle defined in a previous [layout peek](#) -handle command.

### Usage

**\$P file**

### Arguments

None

### Returns

File name for a peek handle.

### Example

This example create a peek handle, returns the layout file name, and then deletes it:

```
set layout [lindex lab2.gds]

# Create peek handle.
set peek [layout peek $layout -handle mypeek]

# Return file name for the peek handle.
$peek file

# Delete the peek handle.
$peek delete
```

### Related Commands

[layout peek](#)

[\\$P delete](#)

[\\$P peek](#)

## \$P peek

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays units, precision, list of cells, list of topcells, list of layers, and other information for a peek handle defined in a previous [layout peek](#) -handle command.

### Usage

**\$P peek** [-bbox *cellname*] [-cblock] [-cblockcount] [-cellcount] [-cells]  
[-child *cellname*] [-lastmodtime] [-layers] [-parent *cellname*] [-precision]  
[-refcount *cellname*] [-topcell] [-topcells] [-type] [-undefcells] [-units]

---

#### Note



Only one -cblock or -cblockcount option can be defined in a \$P peek command. If more are defined, the later option will take precedence.

---

### Arguments

- -bbox | -cblock | -cblockcount | -cellcount | -cells | -child | -lastmodtime | -layers | -parent | -precision | -refcount | -topcell | -topcells | -type | -undefcells | -units

Specifies information to query. If no option is specified, units are displayed.

- bbox — Queries cell's coordinate space for the specified *cellname*.
- cblock — Queries for cblock existence.
- cblockcount — The number of cblocks in this file.
- cellcount — Queries the number of cells in the layout.
- cells — All cell names available in the layout.
- child — A list of children for the specified *cellname*.
- lastmodtime — Queries for creation date from GDS file.

---

#### Note



The command only accepts GDS files.

---

- layers — All layers available in the layout.
- parent — The parent cell for the specified *cellname*.
- precision — A value of 1/units.
- refcount — A count of reference instances to the specified *cellname*.
- topcell — Determines the topcell with the most descendants.

- topcells — All topcells available in the layout.
- type — Specifies the file type, such as GDS, OASIS, or OASIS-STRICT-MODE (for OASIS files in strict mode).
- undefcells — Displays cells referenced but not defined.
- units — Provides units in microns.

## Returns

Returns units, precision, list of cells, list of topcells, list of layers, and other information for a *peek* handle.

- -bbox — Returns cell's coordinate space as {*x y width height*} for the specified *cellname*. *x* and *y* are the coordinates of the lower left corner of the bounding box. The width and height of the bounding box are defined in um.
- -cblock — Returns 1 to indicate existence of one or more cblocks, and 0 otherwise.
- -cblockcount — Returns the number of cblocks in the layout.
- -cellcount — Returns the number of cells in the layout.
- -cells — Returns all cell names available in the layout.
- -child — Returns a list of children for the specified *cellname*.
- -lastmodtime — Returns creation date from the specified GDS file.
- -layers — Returns all layers available in layout, including layers which are defined yet contain no shapes.
- -parent — Returns the parent cell for the specified *cellname*.
- -precision — Returns a value of 1/units.
- -refcount — Returns a count of reference instances to the specified *cellname*.
- -topcell — Returns the topcell with the most descendants.
- -topcells — Returns all topcells available in the layout.
- -type — Returns the file type.
- -undefcells — Returns cells referenced but not defined.
- -units — Returns the units in microns.

If no option is specified, the units are returned.

## Examples

This example create a peek handle, uses it, and then deletes it:

```
% set layout [lindex mult.oas]

# Create peek handle.
% set P [layout peek $layout -handle mypeek]

# Get bounding box for topcells.
```

**\$P peek**

---

```

% array set peekdata [$P peek -type -bbox -topcells]
% set bboxes [$P peek -bbox [lsort -dictionary $peekdata(topcells)]]
% foreach topbbox $bboxes {
%   puts "Boundingbox for topcell [lindex $topbbox 0]: [lindex $topbbox 1]"
% }

% $P peek -cblock
1

% $P peek -cblockcount
2

% $P peek -cellcount
9

% $P peek -cells
r1220 r1230 r1240 r1310 r1620 r1720 r2310 r3500 mix

% $P peek -child mix
{mix {r3500 r1620 r1220 r2310 r1720 r1230 r1240 r1310}}
% $P peek -child r3500
{r3500 {}}

% $P peek -layers
{1 0} {2 0} {4 0} {5 0} {6 0} {7 0} {8 0} {9 0} {10 0} {28 0} {29 0} {30 0}

% $P peek -parent r3500
{r3500 mix}
% $P peek -parent mix
{mix {}}

% $P peek -precision
1000.0

% $P peek -refcount r3500
{r3500 122}
% $P peek -refcount mix
{mix 0}

% $P peek -topcell
mix

% $P peek -topcells
mix add1 add2

% $P peek -type
OASIS

% $P peek -undefcells
% r1230

% $P peek -units
0.001

# Delete the peek handle, preserving cache for subsequent runs.
% $P delete

```

## Related Commands

[layout peek](#)

[\\$P delete](#)

[\\$P file](#)

## Overlay Object

The Overlay object is a collection of one or more superimposed layouts for the purpose of visual comparison. You use overlays to find similarities and differences in your designs. The overlay feature is only used to view layouts. Changing paths or polygons cannot be performed with this feature. However, you can select objects in any of the layouts, take ruler measurements, and use all layout viewing capabilities.

### \$O layouts

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

#### Description

For overlays only, displays all layouts within the overlay.

#### Usage

**\$O layouts**

#### Arguments

None.

#### Returns

Returns all layouts within the overlay.

#### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
DESIGNrev GUI > Layouts pulldown > overlay0

% set O [$wb cget -__layout]
overlay0
DESIGNrev GUI > Layouts pulldown > layout0

% set lay [$wb cget -__layout]
layout0
% $O layouts
L1 L2
% $lay layouts
Error: layout0 layouts: layout handle is not an overlay
```

#### Related Commands

[\\$O layoutHandle](#)

[\\$O overlayout](#)

[\\$L isOverlay](#)

[\\$L layerFilters](#)

[\\$O overlayCells](#)

[layout overlays](#)

[\\$L isReferenced](#)

## \$O layoutHandle

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays the layout handle, *layoutM*, for the specified overlay layout Id, *LN*.

### Usage

**\$O layoutHandle *overlayLayoutId***

### Arguments

- ***overlayLayoutId***

A required argument identifying an overlay Id to return the layout handle for.

### Returns

Returns layout handle, *layoutM*, for the specified overlay layout Id, *LN*.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
```

**DESIGNrev GUI > Layouts pulldown > overlay0**

```
% set O [$wb cget -__layout]
overlay0
```

**DESIGNrev GUI > Layouts pulldown > layout0**

```
% set lay [$wb cget -__layout]
layout0
% $O layoutHandle L1
layout0
% $lay layoutHandle L1
Error: layout0 layoutHandle: layout0 is not an overlay
```

### Related Commands

[\\$O layouts](#)

[\\$O overlayout](#)

[\\$L isOverlay](#)

[\\$L layerFilters](#)

[\\$O overlayCells](#)

[layout overlays](#)

[\\$L isReferenced](#)

## \$O overlayCells

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

For overlays only, returns all overlaid cells in the overlay. Optionally, returns the overlaid cells for just the layout specified by the overlay layout Id.

### Usage

**\$O overlayCells** [-layout *overlayLayoutId*]

### Arguments

- -layout *overlayLayoutId*  
Return the overlaid cells for only the layout specified by the overlay layout Id.

### Returns

Returns overlaid cells in the overlay.

### Example

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0
```

**DESIGNrev GUI > Layouts pulldown > overlay0**

```
% set O [$wb cget -_layout]  
overlay0
```

**DESIGNrev GUI > Layouts pulldown > layout0**

```
% set lay [$wb cget -_layout]  
layout0  
% $O overlayCells  
L2-lab1 L1-lab1  
% $O overlayCells -layout L1  
L1-lab1  
% $lay overlayCells  
Error: layout0 overlayCells: layout handle is not an overlay
```

### Related Commands

[\\$O layoutHandle](#)

[\\$O layouts](#)

[\\$O overlayout](#)

[layout overlays](#)

[\\$L isOverlay](#)

[\\$L isReferenced](#)

[\\$L layerFilters](#)



## \$O overlayout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

For overlays only, saves the overlay as an ASCII setup data file.

### Usage

**\$O overlayout** [-layout *overlayLayoutId*]

### Arguments

- *filename*  
Name of the ASCII setup data file.

### Returns

None.

### Example

**DESIGNrev GUI > Layouts pulldown > overlay0**

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set O [$wb cget -_layout]
overlay0
% $O overlayout setup.ovly
% ls
setup.ovly
```

### Related Commands

[\\$O layoutHandle](#)

[\\$O overlayCells](#)

[\\$L isOverlay](#)

[\\$L layerFilters](#)

[\\$O layouts](#)

[layout overlays](#)

[\\$L isReferenced](#)

## 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.

#### Returns

None.

#### Example

None.

#### Related Commands

[Macros delete](#)

[Macros menu](#)

[Macros show](#)

## 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.

### Returns

None.

### Example

None.

### Related Commands

[Macros create](#)

[Macros menu](#)

[Macros show](#)

## 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**

### Arguments

None.

### Returns

None.

### Example

None.

### Related Commands

[Macros create](#)

[Macros delete](#)

[Macros show](#)

## Macros show

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays information about the macros currently registered with the application.

### Usage

**Macros show** [*macroName*]

### Arguments

- *macroName*  
An optional argument specifying the name of the macro about which you are requesting information.

### Returns

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-in procedure.

### Example

None.

### Related Commands

[Macros create](#)

[Macros delete](#)

[Macros menu](#)

## 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 follow, the variable `$str` always refers to the handle of an actual StringFeature object.

---

## StringFeature

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Creates a string feature.

### 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. Coordinates are in *database units* (dbu).
- **w h**  
Required arguments defining the width and height of the bounding box for the StringFeature. Coordinate lengths are in *microns*.

- *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. Cannot include newline characters.
- *justify*  
Required argument defining how the string is justified in the bounding box. Allowed values are:
  - l — left
  - r — right
  - c — center

## Returns

The handle of the stringfeature.

## 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 update the viewer to see the new object.

## Related Commands

[\\$str addToLayout](#)

## \$str addToLayout

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Inserts a stringfeature into an existing layout. 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 database units (dbu).
- ***layer***  
The layer on which the StringFeature is to be added.

### Returns

None.

### Example

None.

### Related Commands

[StringFeature](#)



## 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 no commands for creating them. The **cwbLayoutView object methods** let you solicit information about what is displayed there.

---


**i** 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.

---

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.

---

**Note**

 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

Displays the name of the currently viewed cell.

### Usage

**\$V cell**

### Arguments

None.

### Returns

Returns the name of the currently viewed cell.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V cell
a2311
```

### Related Commands

None.

## \$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 (dbu) you want to have converted into screen units.

### Returns

The X and Y coordinates expressed in screen units.

### Example

```
% set wb [find objects -class cwbWorkBench]  
::cwbWorkBench::cwbWorkBench0  
% set V [$wb cget -_layoutView]  
::cwbLayoutView0  
% $V databaseToScreenUnits 40489 -39355  
300.001055375 600.000661386
```

### Related Commands

[\\$V micronToScreenUnits](#)

[\\$V screenToDatabaseUnits](#)

[\\$V screenToMicrons](#)

## \$V depth

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays the current view depth.

### Usage

**\$V depth**

### Arguments

None.

### Returns

Returns the current view depth.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V depth
4
```

### Related Commands

None.

## \$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, which can include a directory path as part of the filename. If a file already exists by that name in the destination directory, it is overwritten.

### Returns

None.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V exportView gif a1720.gif
% ls
a1720.gif
...
```

### Related Commands

None.

## \$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 you want to have converted into screen units.

### Returns

The X and Y coordinates expressed in screen units.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V micronToScreenUnits 300 600
1037.68336634 -1217.42085032
```

### Related Commands

[\\$V databaseToScreenUnits](#)

[\\$V screenToDatabaseUnits](#)

[\\$V screenToMicrons](#)

## \$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 (dbu).

### Returns

The X and Y coordinates expressed in database units.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V screenToDatabaseUnits 300 600
40489 -39355
```

### Related Commands

[\\$V databaseToScreenUnits](#)

[\\$V micronToScreenUnits](#)

[\\$V screenToMicrons](#)

## \$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.

### Example

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V screenToMicrons 1037.68336634 -1217.42085032
300.0 600.0
```

### Related Commands

[\\$V databaseToScreenUnits](#)

[\\$V micronToScreenUnits](#)

[\\$V screenToDatabaseUnits](#)

## version

Tools Supported: Calibre DESIGNrev, WORKbench, LITHOview, MDPview

### Description

Displays the product version string. It can be used in Tcl scripts and at the Tcl prompt to return the product version string.

### Usage

**version**

### Arguments

None.

### Returns

Returns the product version string.

### Example

```
% version  
2011.1_0.12
```

### Related Commands

None.



## Overview

Typically, the Calibre layout viewers are accessed using the Calibre DESIGNrev GUI. Alternately, layout viewer batch commands can be used. Additionally, these commands can be combined into stand-alone programs called scripts. Scripts are a simple way to string together a series of batch commands for execution together. It is often useful to develop a script interactively, to see the output of each command as you step through them.

This chapter gives an example of writing a script to output layer and cell data about a layout, and highlighting cells that are empty.

The batch commands for the Calibre DESIGNrev, WORKbench, LITHOview, and MDPview layout viewers are Tcl-based commands, providing access to internal application functions. Executing batch commands for Calibre layout viewers 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.

This chapter shows you how to use various DESIGNrev batch commands, access information in a layout database, and write layout information to a file. Finally, we combine the commands into a script.

This example of looping through a layout database is a teaching aid for writing scripts rather than a concise method for extracting database information. To rapidly extract information about a layout file without loading it, use the [layout peek](#) command.

This chapter covers the following information:

<b>Opening a Data File for Writing.....</b>	<b>338</b>
<b>Determining the Layers and Topcell .....</b>	<b>339</b>
<b>Writing a Procedure .....</b>	<b>339</b>
<b>Opening a Data File for Viewing .....</b>	<b>341</b>
<b>Review of Executed Commands .....</b>	<b>342</b>
<b>Creating a Batch Script.....</b>	<b>343</b>
<b>Creating a Macro.....</b>	<b>347</b>

## Opening a Data File for Writing

DESIGNrev writes batch command results to the terminal window, where your results can be studied. However, because extracting information from a layout database often involves several commands, results can be spread out and difficult to analyze. Writing to a file solves this problem by allowing you to view your results without any extraneous information.

First, invoke DESIGNrev in GUI mode, and load in a layout of your choosing:

```
calibredrv -m mylayout.gds
```

From the terminal window, press **Enter** to access the Tcl “%” prompt.

Next, open a data file to which we will write our results:

```
set fileID [open myoutput.txt w]
```

The Tcl command, **set**, tells Tcl to define a new variable “fileID” and to set its value. Tcl recognizes the square brackets as indicating the need for command substitution to determine the value to place in the “fileID” variable.

The Tcl command, **open**, opens a file and returns the file ID it uses to communicate with that file. The argument “w” tells Tcl to open the file as write-only. Tcl will create the file if it does not exist and overwrite it if it does exist.

Get the layout handle:

```
set cwb [find objects -class cwbWorkBench]
set mylayout [$cwb cget -_layout]
```

It is a good practice to save the handles as variables for any object you may need to manipulate. If you assume the layout handle is “layout0” instead of obtaining the handle as in the previous two Tcl commands, you run the risk of operating on the incorrect layout database.

Now, write a line of text to the data file:

```
puts $fileID "This file displays layer and cell information about the\
[$mylayout file] database."
```

The **puts** command writes to the opened file. The command takes two arguments, the file ID (where to print the information) and the data (what to add to the file). Using the file ID as the first argument instructs the command to write the data to the file rather than the terminal window. The string enclosed in quotes is the data to write to the file. The quotes tell the command to treat all of the words in the string as a single object.

Though there are quotes around the string, Tcl recognizes the square brackets as indicating the need for command substitution and the “\$” as indicating a need for variable substitution. The instance command, “\$mylayout file”, is a layout viewer batch command which returns the name of the layout file.

## Determining the Layers and Topcell

The following command shows you how to write the database layers to the data file.

```
puts $fileID "Layers: [$mylayout layers]"
```

As in the previous **puts** command, the string enclosed in quotes is the data written to the file. However, this string is more complex. This **puts** command writes the string “Layers:” followed by the value or values returned by processing the instance command, “\$mylayout layers”. This layout viewer command returns the list of the layers in the database.

Write the topcell of the layout to the file. Topcells are cells with no references to them.

```
puts $fileID "Topcell: [$mylayout topcell]"
```

This puts statement is much like the previous one, except that it calls a different instance command, “\$mylayout topcell”, which returns the name of the topcell. We also have the option of creating a variable to hold the name of the topcell returned by the instance command.

```
puts $fileID "Topcell: [set top [$mylayout topcell]]"
```

This string is more complex. The **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, “\$mylayout topcell”.

---

### Note



If you want to print special characters such as “[” “]” or “\$”, you can do so by preceding them with a backslash “\”. In addition, “\n” represents a newline character, and “\t” represents a tab character.

---

## Writing a Procedure

We will now write a Tcl procedure to print out the children of a given cell, and we will invoke the procedure at the end of this section. You define a Tcl procedure by specifying what it is called, the arguments it takes, and what it does.

As we do not know the number of levels of cells in the layout database, we need to use either a recursive or an iterative procedure to navigate the cell hierarchy. We’ve chosen recursion, as it is generally more compact and useful when considering objects with a complex nested list structure.

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

Here is our procedure. Type or copy & paste it in:

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

Procedures always take the form:

```
proc <name> <arguments> <body>
```

Where the arguments are as follows:

- **proc** is a Tcl command that creates a procedure.
- The *name* of our procedure is “*proc\_cell\_tree*”.
- The *arguments* are always enclosed in braces ({ }). This argument list contains four arguments, and within the body of the procedure, these arguments are used as variables:

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

- 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. You can think of a procedure as a mini-script. It contains any commands needed to perform its processing.

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

The body of the procedure first checks to see if you passed it a cell name. If you did not, the value of C will be “”, which is the default value. In this case, the procedure extracts the cell name using the “\$mylayout 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 the procedure does is cycle through the children of the cell, invoking itself to print their names and children.

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

The **foreach** command creates a *loop variable* (child) 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.

Invoke the previously-created procedure to write the children of the topcell to the file:

```
proc_cell_tree $mylayout $fileID
```

## Opening a Data File for Viewing

Close the data file to which we wrote our results, to save what we have written to the file:

```
close $fileID
```

Reopen the results file, so you can view its contents:

```
set fileID [open myoutput.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", which tells Tcl you want to *read* the data in the file.

The Tcl command, **read**, requires at least one argument, the file ID (where to find the information). It returns the information it reads to the application. The layout viewer prints the return values for all commands to the shell, displaying it for you.

Close the file when you are done with it:

```
close $fileID
```

You must always close any file you open to avoid data corruption and system resource depletion.

## Review of Executed Commands

Here's a transcript of the commands for writing layer and cell information to a results file. This should match what you have typed in:

```
/labs/drvbatch % ls
mylayout.gds

/labs/drvbatch % calibredrv -m mylayout.gds

// Calibre DESIGNrev v2010.3 () Thu Aug  6 20:17:12 PDT 2010
//
...

% ls
mylayout.gds

% set fileID [open myoutput.txt w]
file9

% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0

% set mylayout [$cwb cget -_layout]
layout0

% puts $fileID "This file displays layer and cell information about\
the [$mylayout file] database."

% puts $fileID "Layers: [$mylayout layers]"

% puts $fileID "Topcell: [$mylayout topcell]"

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

% proc_cell_tree $mylayout $fileID

% close $fileID

% set fileID [open myoutput.txt r]
file9

% read $fileID
This file displays layer and cell information about the
mylayout.gds database.
Layers: 7 8 9 10 11 30 31 1 2 3 4 5 6
Topcell: mp_core
```

```
--> mp_core
--> AOI32X4
--> aoI32_4xaww3wwwwwbhkbsly
--> XNOR2
--> AOI2BB2X1
--> OAI32X4
--> oai32_4xaww3wwwwwtDKunJy
--> AOI222X1
--> aoI222_1xawwpwwwwwl1f2fcy
--> OAI31X1
--> oai31_1xawwewwwwwukxutty
--> AOI21X2
--> aoI21_2xaww9wwwwwcvsIymy
...

% close $fileID

% exit

/labs/drvbatch % ls
mylayout.gds
myoutput.txt
```

## Creating a Batch Script

Tcl scripts are stand-alone programs that you write using the Tcl language, plus any language extensions you have available. Calibre layout viewer scripts are Tcl scripts written using the Calibre layout viewer Tcl extensions and batch commands.

We will run our layout viewer script in batch mode, by including the pathname for the script in the DESIGNrev invocation. We will also add arguments to the script for the layout database pathname and the data output filename:

```
calibredrv myscript.tcl mylayout.gds myoutput.txt
```

The code in the previous section will be expanded to include an example of writing a recursive procedure that checks whether a cell has contents. The resulting script outputs layer and cell data about a layout, and it highlights whether any particular cell is empty.

In addition to adding a new Tcl procedure, the commands we typed into the console in the previous section have been modified to convert them to a script. Here is an example of the batch commands we have discussed, placed into a script format:

```
# Filename: myscript.tcl

# Description: Display layer and cell information about a database.

# This script accepts two optional arguments:
#   arg 0 --> layout filename
#   arg 1 --> output filename

# Default filename argument values.
set layoutfilename "mylayout.gds"
```

```

set outputfilename "myoutput.txt"

# Print out the children of a given cell.
proc proc_cell_tree { L F {C ""} {Indent ""} } {
    if { $C == "" } {
        set C [$L topcell]
    }
    if { [ check_cell_empty $L $C ] } {
        puts $F "$Indent--> $C -- Empty cell"
    } else {
        puts $F "$Indent--> $C"
    }
    append Indent "\t"
    foreach child [$L children $C] {
        proc_cell_tree $L $F $child $Indent
    }
}

# Check for empty cells.
proc check_cell_empty { L cell_name } {
    if { [ $L exists cell $cell_name ] } {
        # Check if cell contains at least one polygon, wire, or text.
        foreach layer [ $L layers ] {
            if { [ $L iterator count poly $cell_name $layer ] != 0 || \
                [ $L iterator count wire $cell_name $layer ] != 0 || \
                [ $L iterator count text $cell_name $layer ] != 0 } {
                return 0
            }
        }
        # If cell only contains a reference, check if reference is empty.
        if { [ $L iterator count ref $cell_name ] != 0 } {
            set ref_list [ $L iterator ref $cell_name range 0 end ]
            foreach ref $ref_list {
                set refname [ lindex [ split $ref ] 0 ]
                if { ![ check_cell_empty $L $refname ] } {
                    return 0
                }
            }
        }
        return 1
    }
}

# Argument processing.
if { $argc == 0 } {
    puts "$argv0: Default layout filename: $layoutfilename"
    puts "$argv0: Default output filename: $outputfilename"
} elseif { $argc == 1 } {
    set layoutfilename [ lindex $argv 0 ]
    puts "$argv0: Default output filename: $outputfilename"
} elseif { $argc == 2 } {
    set layoutfilename [ lindex $argv 0 ]
    set outputfilename [ lindex $argv 1 ]
} else {
    puts "$argv0: Wrong number of arguments, $argc."
}

```



```

    puts "Usage: $argv0 [layout filename] [output filename]"
    puts "$argv0: Default layout filename: $layoutfilename"
    puts "$argv0: Default output filename: $outputfilename"
    exit
}
if { ![ file exists $layoutfilename] } {
    puts "$argv0: Database $layoutfilename does not exist"
    exit
}

# Open data file for writing.
set fileID [open $outputfilename w]

# Capture layout handle.
set mylayout [layout create $layoutfilename]

# Output layer and cell information to data file.
puts $fileID "This file displays layer and cell information about\
the [$mylayout file] database."
puts $fileID "\nLayers:"
foreach lay [lsort -integer [$mylayout layers]] {
    puts $fileID "--> $lay"
}
puts $fileID "\nCells:"

# Write out results to data file.
proc_cell_tree $mylayout $fileID

close $fileID
set fileID [open $outputfilename r]
read $fileID
close $fileID

```

Here is the transcript from running this batch script:

```

/labs/drvbatch % calibredrv myscript.tcl mylayout.gds myoutput.txt
Collecting data...
Analyzing data...
Sorting data...

```

----- ----- LAYOUT FILE OBJECT SUMMARY ----- -----	
Type	Count
LAYER	13
CELL	149
PLACEMENT	135112
ARRAY	0
RECTANGLE	131797
POLYGON	1440
PATH	0
TEXT	0
PROPERTY	0

```
-----  
-----                                LOAD LAYOUT SUMMARY                                -----  
-----  
  
GDS FILE READ FROM '/labs/drvbatch/mylayout.gds'  
  
DATABASE PRECISION:  0.001 user units per database unit  
PHYSICAL PRECISION:  1e-09 meters per database unit  
  
LAYOUT FILE SIZE =           12382 K =           12 M  
LAYOUT MEMORY    =           6659 K =            6 M  
LAYOUT READ TIME = 1 sec REAL.  TOTAL TIME = 2 sec REAL.  
  
/labs/drvbatch %
```

Output from this script is as follows:

This file displays layer and cell information about the mylayout.gds database.

Layers:

```
--> 1  
--> 2  
--> 3  
--> 4  
--> 5  
--> 6  
--> 7  
--> 8  
--> 9  
--> 10  
--> 11  
--> 30  
--> 31
```

Cells:

```
--> mp_core  
--> AOI32X4  
-->   aoi32_4xaww3wwwwbhkbsly  
--> XNOR2  
--> AOI2BB2X1  
--> OAI32X4  
-->   oai32_4xaww3wwwwtstkunjy  
--> AOI222X1  
-->   aoi222_1xaww3wwwwl1f2fcy -- Empty cell  
--> OAI31X1  
-->   oai31_1xaww3wwwwukxutty  
--> AOI21X2  
-->   aoi21_2xaww3wwwwcvsimyy  
...
```

## Creating a Macro

You can add new functionality to the layout viewer by writing application extensions, called macros. Once you create a macro, it is available through the Macros menu. By default, 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 section gives a brief example of how to wrap a layout viewer Tcl script with commands to convert it to a macro, and to also make that macro accessible from the layout viewer Macros menu. See the [Writing Macros](#) chapter for detailed information on creating macros.

Assume you have the following simple Tcl procedure which you want made into a macro:

```
# Example Tcl script which sets some preferred key bindings.
proc return_keys { } {
    set cwb [find objects -class cwbWorkBench]
    $cwb bindKey <Key-F2> cwbHelp Help "Open User Manual"
    # additional key bind changes...
}
```

The following Tcl script changes expand the original Tcl procedure into a macro. These changes are discussed at length in the [Writing Macros](#) chapter:

```
# Example macro which sets some preferred key bindings.
proc return_keys { } {
    set cwb [find objects -class cwbWorkBench]
    $cwb bindKey <Key-F2> cwbHelp Help "Open User Manual"
    # additional key bind changes...
}

proc return_keys_plug_in { wbHandle window } {
    return_keys
}

Macros create "my key bindings" return_keys return_keys_plug_in
```

To add the macro to the layout viewer Macros menu, **source** the macro from your .signaext file, and then invoke the layout viewer as always:

```
echo "source macro_file.tcl" >> ~/.signaext
calibredrv
```



## — Symbols —

- .signaext, [93](#), [94](#)
- \$cwb bindKey command, [111](#)
- \$cwb cget -\_layout command, [117](#)
- \$cwb cget -\_layoutView command, [118](#)
- \$cwb cget -\_mainMenus command, [119](#)
- \$cwb getMenuPath command, [121](#)
- \$cwb getRulerFont command, [122](#)
- \$cwb layoutDeleteClbk command, [120](#)
- \$cwb loadDefaultLayerProperties command, [126](#)
- \$cwb loadLayerProperties command, [127](#)
- \$cwb rulerMeasurements command, [129](#)
- \$cwb runMultiRveOnRveOutput command, [131](#)
- \$cwb runRveOnRveOutput command, [132](#)
- \$cwb selectionCloneToNewLayout command, [133](#)
- \$cwb setDepth command, [134](#)
- \$cwb setRulerFont command, [135](#)
- \$cwb showWithLayerFilterClbk command, [137](#)
- \$cwb updateDisplay command, [138](#)
- \$cwb viewedLayoutClbk command, [139](#)
- \$cwb zoomAllClbk command, [140](#)
- \$cwb zoomToRectClbk command, [141](#)
- \$L allowupdates command, [174](#)
- \$L ancestors batch command, [176](#)
- \$L AND batch command, [177](#)
- \$L bbox batch command, [179](#)
- \$L cellname batch command, [180](#)
- \$L cells batch command, [181](#), [224](#)
- \$L children batch command, [182](#)
- \$L clips batch command, [183](#)
- \$L clipsout batch command, [184](#)
- \$L connect batch command, [186](#)
- \$L COPY batch command, [188](#)
- \$L COPYCELL GEOM batch command, [189](#)
- \$L create cell batch command, [190](#)
- \$L create clip batch command, [192](#)
- \$L create layer batch command, [195](#)
- \$L create ref batch command, [198](#)
- \$L create text batch command, [201](#)
- \$L create wire batch command, [203](#)
- \$L customLayerDrawOrder batch command, [205](#)
- \$L delete cell batch command, [206](#)
- \$L delete clip batch command, [207](#)
- \$L delete duplicate batch command, [208](#)
- \$L delete layer batch command, [209](#)
- \$L delete polygon batch command, [210](#)
- \$L delete polygons batch command, [212](#)
- \$L delete ref batch command, [213](#)
- \$L delete wire batch command, [218](#)
- \$L disconnect batch command, [220](#)
- \$L duplicate cell batch command, [222](#)
- \$L exists cells batch command, [223](#)
- \$L expand cell batch command, [225](#)
- \$L expand ref batch command, [227](#), [231](#)
- \$L file batch command, [228](#)
- \$L flatten cell batch command, [229](#)
- \$L gdsout batch command, [233](#)
- \$L gridsnap batch command, [236](#)
- \$L holdupdates command, [238](#)
- \$L import batch command, [240](#)
- \$L instancedbout batch command, [242](#)
- \$L isLayerEmpty batch command, [243](#)
- \$L ismodified batch command, [244](#)
- \$L isOverlay batch command, [245](#)
- \$L isReferenced batch command, [246](#)
- \$L iterator count batch command, [247](#), [252](#), [256](#), [257](#)
- \$L layerconfigure batch command, [258](#)
- \$L layerFilters batch command, [260](#)
- \$L layernames batch command, [262](#)
- \$L layers batch command, [264](#)
- \$L libname batch command, [267](#)
- \$L MASK\_LAYER\_INFO batch command, [269](#)

\$L maxdepth batch command, 268  
 \$L modify origin batch command, 270  
 \$L modify text batch command, 271  
 \$L NOT batch command, 273  
 \$L oasisout batch command, 274  
 \$L OR batch command, 279  
 \$L pushmarkers batch command, 280  
 \$L query batch command, 282  
 \$L rdbout command, 288, 291  
 \$L readonly command, 289  
 \$L refcount command, 290  
 \$L scale batch command, 292  
 \$L SIZE batch command, 293  
 \$L srefsFromAref command, 295  
 \$L textout batch command, 297  
 \$L topcell batch command, 298  
 \$L transcript batch command, 299  
 \$L transcript OUTPUT command, 300  
 \$L transcript RANGE, 301  
 \$L transcript STATE command, 302  
 \$L units database command, 303  
 \$L units microns command, 304  
 \$L units user command, 306  
 \$L update, 308  
 \$L viacell batch command, 309  
 \$L XOR command, 310  
 \$str, 326  
 \$str addToLayout command, 328  
 \$V, 329  
 \$V cell command, 329  
 \$V databaseToScreenUnits commands, 330  
 \$V depth command, 331  
 \$V exportView command, 332  
 \$V micronToScreenUnits command, 333  
 \$V screenToDatabaseUnits command, 334  
 \$V screenToMicrons command, 335

## — A —

addToLayout command, 328  
 Ancestors batch command, 176  
 AND batch command, 177  
 Append  
     on merge, 161  
 asciiout batch command, 178

## — B —

Bbox batch command, 179  
 bindKey command, 111  
 Brackets in Tcl, 39

## — C —

Calibre  
     WORKbench, 70  
 Cell reference  
     expand, 227, 231  
     flattening, 227, 231  
 Cellname batch command, 180  
 Cells batch command, 181  
 cget  
     -\_layout command, 117  
     -\_layoutView command, 118  
     -\_mainMenus command, 119  
 Children batch command, 182  
 Clips  
     batch command, 183  
 Clipsout  
     batch command, 184  
 Command substitution, 39, 42  
 connect batch command, 186  
 COPY batch command, 188  
 COPYCELL GEOM batch command, 189  
 Create Cell batch command, 190  
 Create clip  
     batch command, 192  
 Create Layer batch command, 195  
 Create Polygon batch command, 196  
 Create Ref batch command, 198  
 Create Text batch command, 201  
 Create Wire batch command, 203  
 customLayerDrawOrder batch command, 205  
 cwbLayoutView class, 329  
 cwbWorkBench class, 110

## — D —

Database units, 303  
 Datatypes, 150, 233  
 Delete Cell batch command, 206  
 Delete Clip batch command, 207  
 Delete Duplicate batch command, 208  
 Delete Layer batch command, 209  
 Delete Polygon batch command, 210

Delete Polygons batch command, 212  
Delete Ref batch command, 213  
Delete Wire batch command, 218  
DESIGNrev  
    as background process, 29  
disconnect batch command, 220  
DRC EXTERNAL spacing check, 94  
Duplicate cell batch command, 222

## — E —

Error handling, 63  
Exists cell batch command, 223  
Exists layer batch command, 224  
Expand cell batch command, 225  
Expand ref batch command, 227, 231

## — F —

File batch command, 228  
Flatten cell batch command, 229  
Force\_rename  
    on merge, 162

## — G —

Gdsout batch command, 233  
getRulerFont command, 122  
Grid  
    snapping for layouts, 236  
Gridsnap batch command, 236

## — H —

Handle, 39  
Handles, 24

## — I —

Import Layout batch command, 240  
Instancedbout batch command, 242  
IsLayerEmpty batch command, 243  
Ismodified batch command, 244  
IsOverlay batch command, 245  
IsReferenced batch command, 246  
Iterator Count batch command, 247, 252, 256, 257

## — L —

Layer colors and properties, 100  
layerFilters batch command, 260  
Layernames batch command, 262, 264

Layers batch command, 264  
layout  
    object methods, 142  
    scale, 292  
Layout All batch command, 142  
layout assembly, 58  
Layout Copy batch command, 144  
Layout Create batch command, 149  
layout delete batch command, 154  
layout droasis batch command, 155  
Layout Handles  
    returning, 142  
layout mapping, 152  
Layout Merge batch command, 161  
Layout Overlays ASCII batch command, 165  
Layout peek command, 166  
layoutDeleteClbk command, 120  
LayoutHandle batch command, 319  
Layouts batch command, 318  
LayoutSetDefaultColors batch command, 171  
LayoutType batch command, 266  
loadDefaultLayerProperties command, 126  
Loading  
    shared libraries, 38  
loadLayerProperties command, 127

## — M —

macro  
    extension, 94  
Macro Plug-in, 93  
Macros create command, 322  
Macros menu command, 324  
MAPI functions, 108  
Modify Origin batch command, 270  
Modify Text batch command, 271

## — N —

NOT batch command, 273

## — O —

Oasisout batch command, 274  
OR batch command, 279  
overlayCells batch command, 320  
overlayout batch command, 321  
Overwrite  
    on merge, 162

## — P —

Parent, [42](#)  
 Peek delete batch command, [311](#)  
 Peek file batch command, [313](#)  
 Peek peek batch command, [314](#)  
 Pushmarker  
     batch command, [280](#)

## — Q —

Query batch command, [282](#)

## — R —

rdbout, [288](#), [291](#)  
 Recursion, [51](#)  
 refcount, [290](#)  
 Rename  
     on merge, [162](#)  
 runMultiRveOnRveOutput command, [131](#)  
 runRveOnRveOutput command, [132](#)  
 -rve option, [32](#)

## — S —

Scale  
     batch command, [292](#)  
 Screenscaptures (\$V exportView), [332](#)  
 selectionCloneToNewLayout command, [133](#)  
 setDepth command, [134](#)  
 setRulerFont command, [135](#)  
 Shared libraries, loading, [38](#)  
 shell mode  
     problems, [20](#)  
 showWithLayerFilterClbk command, [137](#)  
 SIZE batch command, [293](#)  
 Square brackets in Tcl, [39](#)  
 StringFeature command, [326](#)

## — T —

Tcl books, [17](#)  
 Tcl commands  
     for database objects, [101](#)  
 Tcl macros, [322](#)  
 Textout batch command, [297](#)  
 topcell, [298](#)  
 Transcript, [300](#)  
 Transcript STATE, [302](#)

## — U —

Units database command, [303](#)  
 Units microns command, [304](#)  
 Units user command, [306](#)  
 updateDisplay command, [138](#)

## — V —

Variable substitution, [42](#)  
 version command, [336](#)  
 viacell batch command, [309](#)  
 viewedLayoutClbk command, [139](#)

## — X —

XOR command, [310](#)

## — Z —

zoomAllClbk command, [140](#)  
 zoomToRectClbk command, [141](#)



# Third-Party Information

For third-party information, refer to *[Third-Party Software for Calibre Products](#)*.



# End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:  
[www.mentor.com/eula](http://www.mentor.com/eula)

## IMPORTANT INFORMATION

**USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S 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 (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer 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 Customer does not agree to these terms and conditions, promptly return or, in the case of Software 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. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, 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. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.
- 1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, 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 Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; 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. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. 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. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product

improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.
4. **BETA CODE.**
  - 4.1. Portions or all of certain 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 Customer 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 Customer's 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.
  - 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
  - 4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees 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 Customer's 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 Subsection 4.3 shall survive termination of this Agreement.
5. **RESTRICTIONS ON USE.**
  - 5.1. Customer 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. Customer 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. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.
  - 5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.
  - 5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. 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/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at <http://supportnet.mentor.com/about/legal/>.
7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.
8. **LIMITED WARRANTY.**
  - 8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products 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. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."
  - 8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.
9. **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 RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE 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 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
11. **INDEMNIFICATION.** CUSTOMER AGREES 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 THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.
12. **INFRINGEMENT.**
  - 12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

- 12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.
- 12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.
- 12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.
13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.
- 13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.
- 13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.
14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.
15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.
16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all 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. The provisions of this Section 17 shall survive the termination of this Agreement.
18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is 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 the courts 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 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. This section shall not

restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **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.
20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. 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. 100615, Part No. 246066