

SIEMENS EDA

Calibre® DESIGNrev™

Reference Manual

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

Software Version 2024.1
Document Revision 25

Unpublished work. © 2024 Siemens

This Documentation contains trade secrets or otherwise confidential information owned by Siemens Industry Software Inc. or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this Documentation is strictly limited as set forth in Customer's applicable agreement(s) with Siemens. This Documentation may not be copied, distributed, or otherwise disclosed by Customer without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

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

No representation or other affirmation of fact contained in this Documentation shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

If you have a signed license agreement with Siemens for the product with which this Documentation will be used, your use of this Documentation is subject to the scope of license and the software protection and security provisions of that agreement. If you do not have such a signed license agreement, your use is subject to the Siemens Universal Customer Agreement, which may be viewed at <https://www.sw.siemens.com/en-US/sw-terms/base/uca/>, as supplemented by the product specific terms which may be viewed at <https://www.sw.siemens.com/en-US/sw-terms/supplements/>.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS DOCUMENTATION INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY. SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS DOCUMENTATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

TRADEMARKS: The trademarks, logos, and service marks (collectively, "Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' Marks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html. The registered trademark Linux[®] is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

About Siemens Digital Industries Software

Siemens Digital Industries Software is a global leader in the growing field of product lifecycle management (PLM), manufacturing operations management (MOM), and electronic design automation (EDA) software, hardware, and services. Siemens works with more than 100,000 customers, leading the digitalization of their planning and manufacturing processes. At Siemens Digital Industries Software, we blur the boundaries between industry domains by integrating the virtual and physical, hardware and software, design and manufacturing worlds. With the rapid pace of innovation, digitalization is no longer tomorrow's idea. We take what the future promises tomorrow and make it real for our customers today. Where today meets tomorrow. Our culture encourages creativity, welcomes fresh thinking and focuses on growth, so our people, our business, and our customers can achieve their full potential.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Revision History

Revision	Changes	Status/ Date
25	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released January 2024
24	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released October 2023
23	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released July 2023
22	Modifications to improve the readability and comprehension of the content. Approved by Lucille Woo. All technical enhancements, changes, and fixes listed in the <i>Calibre Release Notes</i> for this products are reflected in this document. Approved by Michael Buehler.	Released April 2023

Author: In-house procedures and working practices require multiple authors for documents. All associated authors for each topic within this document are tracked within the Siemens EDA documentation source. For specific topic authors, contact the Siemens Digital Industries Software documentation department.

Revision History: Released documents maintain a revision history of up to four revisions. For earlier revision history, refer to earlier releases of documentation which are available on <https://support.sw.siemens.com/>.

Table of Contents

Revision History

Chapter 1

Introduction to Using Tcl With the Calibre Layout Viewers	15
Why Use Tcl?	15
Tcl Versus Tk and What the Distinction Means to You	17
Extensions to Tcl	17
Things to Know About Tcl	19
Tclsh	19
Tcl Syntax	19
Tcl Comments	20
Special Characters	21
Tcl Lists	22
Variables	22
Layout Viewers and Tcl	24
Layout Viewer Objects and Handles	24
Layout Viewer Commands and Object Methods	25
Most Database Objects are not Tcl Objects	25
Layout Viewer Application Extension Files	26
Questions and Answers	26
Syntax Conventions	27

Chapter 2

Getting Started	29
Issuing Tcl Commands	30
Interactive GUI	30
Interactive Shell	32
Batch	33
Batch GUI	34
Command	34
Tcl and Shared Libraries	35
Explorations in Invocation Options	36
Example 1 - Listing the Cells in a Layout Using the Interactive Shell Option	36
Example 2 - Listing the Cells in an Open Layout Using the Interactive GUI Option	38
Example 3 - Listing the Cells in a Layout Using Batch Mode	41
Explorations In Writing Procedures	43
Example 4 - Using a Simple Procedure to Write the Layout Hierarchy	43
Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy	47
Questions and Answers	52

Chapter 3	
Writing Layout Viewer Scripts	53
Executing Tcl Scripts	53
Scripts for Layout Generation and Assembly	54
Script 1: Writing a Script to Generate a New Layout	54
Script 2: Setting up a Script to Accept Passed In Arguments	56
Script 3: Debugging and Adding Error Handling	59
Script 4: Moving a Cell's Origin	64
Script 5: Changing Cell Names in a GDS File	66
Chapter 4	
Writing Macros	71
About Writing Macros	72
Writing Command Procedures	72
Writing GUI Plug-ins	73
Register the Macro	74
Load the Macro	74
Simple Macro Example	75
Explorations in Creating Macros	77
Transforming a Simple Script into a Command Procedure	77
Exploring Ways to Obtain User-Supplied Data	80
Writing a GUI Plug-in Procedure	82
Displaying a Message to the User	83
Adding the Macro to the Menu	85
Building in Error Handling	86
Setting Up Your Macro to Load Automatically	89
Sample Macro Code	89
Macro Examples to Study and Learn	92
List All Layers	92
Draw a Circle	92
DRC External Spacing Check	94
Questions and Answers About Macros	97
Chapter 5	
Layout Viewer Batch Commands and Object Methods	99
Unsupported Commands and Arguments in High Capacity (HC) Mode	100
cwbWorkBench Commands	101
cwbLoadRdbFileInRve	102
cwbOpenDBInRve	103
cwbWorkBench Object Methods	104
\$swb bindKey	106
\$swb cget -_layout	113
\$swb cget -_layoutView	114
\$swb deleteLayoutClbk	115
\$swb exportLayerPalette	116
\$swb getMenuPath	117
\$swb getRulerUnits	118
\$swb getSelectionIterator	119

Table of Contents

\$cwb getSelectionPoint	125
\$cwb getSelectionRect	126
\$cwb getVisibleLayers	127
\$cwb hasSelectionPoint	128
\$cwb loadDefaultLayerProperties	129
\$cwb loadLayerProperties	130
\$cwb reloadLayoutClbk	131
\$cwb rulerMeasurements	132
\$cwb runMultiRveOnRveOutput	134
\$cwb runRveOnRveOutput	135
\$cwb selectionCloneToNewLayout	136
\$cwb setDepth	137
\$cwb setLayerVisibility	138
\$cwb setReferenceVisibility	141
\$cwb show	143
\$cwb showWithLayerFilterClbk	144
\$cwb synchronizeWindowView	145
\$cwb unsynchronizeWindowView	146
\$cwb updateDisplay	147
\$cwb viewedLayoutClbk	148
\$cwb waitForDrawing	149
\$cwb zoomAllClbk	150
\$cwb zoomToRectClbk	151
Layout Commands	152
layout all	153
layout copy	154
layout copy2	157
layout create Commands	159
layout create	160
layout create (GDS or OASIS file)	161
layout create (ASCII RDB)	165
layout create (LEF/DEF or OpenAccess)	168
layout create (mapped layout)	170
layout create (multiple layouts)	175
layout createCache	179
layout delete	180
layout droasis	181
layout filemerge	182
layout filtershapes	211
layout merge	216
layout overlays	220
layout peek	221
layout tbmode	233
Layout Object Methods	235
\$L allowupdates	240
\$L ancestors	242
\$L AND	244
\$L asciiout	246
\$L bbox	248

\$L cellname.	250
\$L cells	251
\$L children	252
\$L clips	254
\$L clipsout	256
\$L connect.	258
\$L COPY	261
\$L COPYCELL GEOM	262
\$L create cell.	263
\$L create clip	265
\$L create layer	271
\$L create polygon	272
\$L create ref	274
\$L create text	278
\$L create wire	281
\$L customLayerDrawOrder	284
\$L delete cell.	285
\$L delete clip	286
\$L delete duplicate object.	287
\$L delete duplicate ref	288
\$L delete layer	289
\$L delete objects	290
\$L delete polygon	292
\$L delete polygons	294
\$L delete ref	295
\$L delete text	297
\$L delete wire	301
\$L disconnect	304
\$L duplicate cell	306
\$L exists cell	307
\$L exists layer.	308
\$L expand cell.	309
\$L expand ref	311
\$L exportNet	313
\$L exportView	314
\$L extractNet	316
\$L file	319
\$L flatten cell	320
\$L flatten ref	322
\$L format	324
\$L gdsout	325
\$L gridsnap	330
\$L holdupdates	333
\$L import layout	335
\$L instancedbout.	337
\$L isLayerEmpty	338
\$L ismodified	339
\$L isOverlay	340
\$L isReferenced	341

Table of Contents

\$L iterator (poly wire text)	342
\$L iterator (ref sref aref)	348
\$L iterator count (poly wire text)	356
\$L iterator count (ref sref aref)	357
\$L layerconfigure	360
\$L layerFilters	361
\$L layernames	363
\$L layers	365
\$L layoutType	367
\$L libname	368
\$L loadlayerprops	369
\$L MASK_LAYER_INFO	370
\$L maxdepth	372
\$L modify layer	373
\$L modify origin	374
\$L modify text	376
\$L NOT	379
\$L oasisout	381
\$L options	388
\$L OR	390
\$L property	392
\$L pushmarkers	394
\$L query	398
\$L rdbout	407
\$L readonly	408
\$L refcount	409
\$L report duplicate ref	410
\$L ruler	411
\$L scale	412
\$L SIZE	414
\$L SNAP	415
\$L srefsFromAref	416
\$L stopconnect	418
\$L textout	420
\$L topcell	421
\$L transcript	423
\$L transcript output	424
\$L transcript range	425
\$L transcript state	426
\$L units database	427
\$L units microns	428
\$L units user	430
\$L update	433
\$L viacell	434
\$L XOR	435
Overlay Commands	437
overlay all	438
overlay create	439
overlay delete	442

overlay filemerge	443
Overlay Objects Methods	446
\$O gdsout	447
\$O layoutHandle	448
\$O layouts	449
\$O oasisout	450
\$O overlayCells	451
\$O overlayout	452
Peek Object Methods	453
\$P delete	454
\$P file	455
\$P peek	456
StringFeature Commands	464
StringFeature	465
StringFeature Object Methods	467
\$str addToLayout	468
cwbLayoutView Object Methods	469
\$V cell	470
\$V databaseToScreenUnits	471
\$V depth	472
\$V exportView	473
\$V micronToScreenUnits	474
\$V screenToDatabaseUnits	475
Macros Commands	476
Macros create	477
Macros delete	480
Macros menu	481
Macros show	482
Miscellaneous Batch Commands	483
version	484
getCWBObjects	485
Appendix A	
Example Script	487
Overview	487
Opening a Data File for Writing	488
Determining the Layers and Topcell	489
Writing a Procedure	490
Opening a Data File for Viewing	491
Review of Executed Commands	492
Creating a Batch Script	494
Creating a Macro	498

Index

Third-Party Information

List of Figures

Figure 1-1. Layouts Menu - Handles and Filenames.	24
Figure 2-1. Interactive GUI	31
Figure 2-2. Graphical Representation of a Recursive Procedure.	50
Figure 4-1. Completed Code (write_hier.tcl).	90
Figure 5-1. Added Menu	117
Figure 5-2. Merge and Preserve.	197
Figure 5-3. Layout Filtersshapes.	214
Figure 5-4. Boolean NOT	380
Figure 5-5. Relating User and Database Units to the Design Grid	431

List of Tables

Table 1-1. Tcl Special Characters	21
Table 1-2. Calibre DESIGNrev Handles	24
Table 1-3. Syntax Conventions	27
Table 2-1. Summary of Options	30
Table 5-1. Unsupported Commands and Arguments in HC Mode	100
Table 5-2. cwbWorkBench Command Summary	101
Table 5-3. cwbWorkBench Object Methods Summary	104
Table 5-4. Supported Key and Mouse Wheel Combinations for Custom Key Bindings ...	107
Table 5-5. Actions You Can Bind To Keys	108
Table 5-6. Layout Commands Summary	152
Table 5-7. layout create Commands Summary	159
Table 5-8. OASIS Standard Properties	224
Table 5-9. Layout Object Methods Summary	235
Table 5-10. Overlay Commands Summary	437
Table 5-11. Overlay Object Methods Summary	446
Table 5-12. Peek Object Methods Summary	453
Table 5-13. Table Offset Order	457
Table 5-14. OASIS Properties	458
Table 5-15. String Feature Commands Summary	464
Table 5-16. StringFeature Object Methods Summary	467
Table 5-17. CwbLayoutView Object Methods Summary	469
Table 5-18. Macros Commands Summary	476
Table 5-19. Miscellaneous Batch Commands Summary	483

Chapter 1

Introduction to Using Tcl With the Calibre Layout Viewers

The Calibre layout viewers provide a powerful scripting capability using the Tcl-based batch commands. The phrase *Calibre layout viewers* is used throughout this manual when referring to the family of layout viewer tools which include Calibre DESIGNrev, Calibre WORKbench, Calibre LITHOview, and Calibre MDPview.

Refer to “[Layout Viewer Batch Commands and Object Methods](#)” on page 99 for reference information on the layout viewer batch commands. For information on using the GUI for the different Calibre layout viewers, refer to the appropriate user documentation:

- Calibre DESIGNrev — [Calibre DESIGNrev Layout Viewer User’s Manual](#)
- Calibre MDPview — [Calibre MDPview User’s and Reference Manual](#)
- Calibre LITHOview and Calibre WORKbench — [Calibre WORKbench User’s and Reference Manual](#)

Refer to “[Issuing Tcl Commands](#)” on page 30 for examples of the invocation commands for the different layout viewers.

Why Use Tcl?	15
Tcl Versus Tk and What the Distinction Means to You	17
Extensions to Tcl	17
Things to Know About Tcl	19
Layout Viewers and Tcl	24
Questions and Answers	26
Syntax Conventions	27

Why Use Tcl?

The Calibre layout viewers include a full version of Tcl that you can use to develop custom scripts for automating frequently performed tasks and extending the functionality of the layout view tools.

Why use Tcl?

- *Tcl/Tk is easy to learn.* In fact, Tcl/Tk is considered to be one of the easiest programming languages to learn.

Why Use Tcl?

- *Tcl/Tk is easy to customize.* The ability to customize makes it possible for the Calibre layout viewers 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 Tcl/Tk lets you send data from one program to another.

Note



Tcl is an evolving open source language that is used to implement a number of interfaces in Calibre. Future versions of Tcl that are adopted by Calibre may not run some of the Tcl code that ran under a previous version of Tcl. For this reason, it is important to understand that some maintenance may be required for an older version of Tcl code to work with a new version of Calibre that uses a later version of Tcl.

Tcl Resources

Be sure to take advantage of some of the books and web sites on the Tcl language. These resources provide a starting point in your search for the reference material that works best for you. It is not an endorsement of any book or web site. 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 Developers Xchange:
www.tcl.tk/scripting/
- TclTutor:
www.tcl.tk/scripting/
- Beginning Tcl:
wiki.tcl.tk/298
- The Tccler's Wiki:
wiki.tcl.tk

- ActiveState.com:
www.activestate.com/Products/ActiveTcl


Tcl Versus Tk and What the Distinction Means to You

Tcl is rarely mentioned without Tk because of the close relationship between the two. Tcl is the language used to write scripts, while Tk is the toolkit of building blocks used to build graphical user interfaces for Tcl programs. Tk is basically an extension to Tcl.

Many people who write scripts using the Calibre layout viewer batch commands never bother to create a graphical user interface for their scripts. If you are one of those people, you do 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 to the proper binary files, libraries, and necessary extensions.
- You can use Tcl without Tk.
- You cannot use Tk without Tcl. Hence, you often see the Tk toolkit written as Tcl/Tk.
- The Calibre layout viewers load everything you need to successfully execute Tcl commands.
- If you run one of the layout viewer 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 package is not loaded, the tool aborts with an error.

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.

The Calibre layout viewers mainly use iTcl/iTk (also referred to as Incr Tcl). iTcl/iTk is an extension to Tcl providing object-oriented programming capabilities. You can learn more about iTcl/itk at:

<http://incrtcl.sourceforge.net/itcl/>

Because the Calibre layout viewers 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. These 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 dependent 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 to Know About Tcl

When working with Tcl, there are some basic concepts you should be familiar with. These include Tcl syntax, characters with special meaning, comments, and variables.

Tclsh	19
Tcl Syntax	19
Tcl Comments	20
Special Characters	21
Tcl Lists	22
Variables	22

Tclsh

Tclsh is a shell-like application that reads Tcl commands from standard input or from a file and evaluates them. The `tclsh` application is useful for becoming familiar with Tcl syntax and trying out Tcl commands.

The default installation location for `tclsh` is `/usr/bin`. You can invoke Tclsh interactively by entering the following:

```
%/usr/bin/tclsh
```

When `tclsh` is invoked interactively it normally prompts for each command with “%”. You can enter Tcl commands at the prompt (standard input) and `tclsh` prints command results and error messages to standard output.

When you invoke `tclsh` interactively, it does not recognize the layout viewer batch commands. In order to execute the batch commands, you must first invoke a layout viewer. Tclsh runs in the window from which you invoke the layout viewer. From this window, you can execute both Tcl commands and layout viewer batch commands.

Tcl Syntax

Tcl is an easy to learn, simple programming language. There are some basic syntax conventions to follow when writing Tcl scripts.

- A Tcl script is composed of commands that are separated by newlines or a ; (semicolon).
- A Tcl command is composed of words that are separated by spaces or tabs. The first word is the name of a command and the rest of the words are arguments to the command.

- Tcl is case sensitive as shown in this example using the Tcl **puts** command:

```
% puts HelloWorld!  
HelloWorld!  
% Puts HelloWorld!  
invalid command name "Puts"
```

The core set of Tcl commands are all lowercase, so when in doubt, use lowercase. The reference pages in “[Layout Viewer Batch Commands and Object Methods](#)” on page 99, display command syntax in the proper case.

Tcl Comments

In Tcl, you use the “#” symbol (referred to as the pound sign or hashtag) to denote the start of a comment. This symbol directs the Tcl compiler to not evaluate the rest of the line.

There are some important points to be aware of when using comments in a Tcl script:

- 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 } {  
    ...  
  }  
}
```

- 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."\  
# -type ok  
  
# and this is also a comment. This one spans \  
multiple lines, even without the # at the beginning \  
of the second and third lines.
```

It is good practice to begin all comment lines with a #.

- 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 is the first character of the command name. Use the semicolon “;” to combine multiple commands into one line and identify where one command ends and the other begins. For example:

```
set myname "John Doe" ;set this_string "next command"  
  
set yourname "Ted Smith" ; #this is a comment
```

Special Characters

In Tcl scripts, some characters have a special meaning.

[Table 1-1](#) describes the special characters used in the examples found in this manual. For a complete list of special characters, refer to one of the resources provided in [“Tcl Resources”](#) on page 16.

Table 1-1. Tcl Special Characters

Characters	Meaning
;	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.
[]	<p>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. For example, you can enter the following to set a variable to the result of processing a command:</p> <pre>set my_layout [layout create sample.gds]</pre> <p>You also use brackets when you want to use variable substitution. For example:</p> <pre>% layout0 iterator wire cellA [list \$layer1 \$layer2] \ range 0 end</pre> <p>Refer to “Tcl Lists” for more information.</p>
{ }	<p>Curly braces instruct the Tcl interpreter to treat the enclosed words as a single string. For example, to create a string containing special characters, such as \$ or \:</p> <pre>set my_string {This book costs \$25.98.}</pre> <p>The Tcl interpreter accepts the string as is, without performing any variable substitution or evaluation. If you want to use variable substitution or evaluation, you should use the Tcl list command and enclose it in brackets. Refer to “Tcl Lists” for more information.</p>

Table 1-1. Tcl Special Characters (cont.)

Characters	Meaning
" "	<p>Quotes instruct the Tcl interpreter to treat the enclosed words as a single string. However, when the Tcl interpreter encounters variables or commands within a string that is in quotes, it evaluates the variables and commands to generate a string. For example, to create a string displaying a final cost calculated by adding two numbers:</p> <pre>set my_string "This book costs \\${expr \$price + \$tax}"</pre>

Tcl Lists

A Tcl list is an ordered collection of numbers, words, strings, or other lists. The Tcl language allows you to create a list in different ways.

For example, you can create a Tcl list using braces as shown here:

```
set my_list { {Item 1} {Item 2} {Item 3} }
```

You can also define a list by using the Tcl list command and enclosing the list in brackets. For example:

```
set my_list [list "Item 1" "Item 2" "Item 3"]
```

If you define variables that are to be used as arguments to a batch command, then you must use the Tcl list command and brackets in order for the variables to be interpreted correctly.

Variables are not interpreted when they are enclosed in braces.

Throughout this manual, braces are used in the batch command usage and argument descriptions to show groupings of arguments. If the braces are enclosed in single quotes, then they are literal braces and should be specified along with the required arguments. Or, you may opt to replace any literal braces with brackets and Tcl list command.

When writing batch scripts that use Tcl lists to specify arguments, it is recommended that you use the Tcl list command (with brackets) instead of using braces. Where applicable, the Tcl list command (with brackets) is used in most examples found in this manual.

Variables

Variables are extremely useful when writing Tcl scripts because they are reusable and context independent. When writing Tcl scripts for a layout viewer, you can use the handle assigned by the layout viewer to refer a specific object. However, handles are system-generated by the layout viewer and are not always intuitive. Instead of using the handle, it is typically easier to assign the handle to a variable because you can choose the variable name and then use it anytime you want to refer to a specific object.

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

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

You use the Tcl **set** command to assign a handle to a variable. The syntax for the set command is:

```
set <variable_name> <value>
```

The first argument is the name you want to assign to the variable and the second argument is the value to assign to the variable name. For example, the following command specifies the variable name as “my_layout” and the assigned value as “layout1”:

```
set my_layout layout1
```

When assigning and using a variable in a script, you prefix a “\$” to the variable to perform variable substitution, which replaces the variable name with the assigned value.

All layout viewer object method commands begin with a variable (for example, \$cwb and \$L). You must assign a handle to this type of variable in order to execute an object method command. For example:

```
set lay [layout create fullchip.oas]  
$lay oasisout new_fullchip.oas
```

This example does the following:

- The `layout create` command creates a layout from *fullchip.oas* and the layout viewer automatically assigns a system-generated handle to the new layout.
- The Tcl **set** command assigns the system-generated handle to the variable “lay”.
- The variable \$lay then replaces \$L in the \$L oasisout command and the system-generated handle is substituted for the variable \$lay.

Refer to “[Layout Viewer Objects and Handles](#)” on page 24 for more information about handles.

Layout Viewers and Tcl

Layout Viewer Objects and Handles	24
Layout Viewer Commands and Object Methods	25
Most Database Objects are not Tcl Objects	25
Layout Viewer Application Extension Files	26

Layout Viewer Objects and Handles

Layout viewer objects include the window, layouts, overlays, and views. When you create or open an object in a layout viewer, the layout viewer assigns a system-generated instance name to the object. This system-generated instance name is referred to as a *handle*. An example of an instance name, or handle, is *layout0*. The layout viewer maintains this handle in memory during the layout viewer session. Each handle is unique and applies to one and only one specific object.

Table 1-2 identifies the built-in handles associated with the different layout viewer objects.

Table 1-2. Calibre DESIGNrev Handles

Object	Handle	Example
window	::cwbWorkBench::cwbWorkBench ¹	::cwbWorkBench::cwbWorkBench0
layouts	layout ⁿ	layout1
overlays	overlay ⁿ	overlay1
views	::cwbLayoutView ⁿ	::cwbLayoutView0

1. *n* is a unique positive number assigned to the object. The layout viewer assigns 0 to the first object that is created in the layout viewer and then increments *n* by 1 with the creation of each successive object of the same type.

The Layouts menu displays the handle names and associated filenames. Figure 1-1 shows an example of the Layouts menu in which three layouts have been created or opened in the layout viewer. Each layout is displayed in the Layouts menu using the format *<handle> : <filename>*. For example, when the file *fullchip.oas* was opened, the layout viewer automatically assigned the handle “layout2” to the file. The string “{No Associated File}” shown in Figure 1-1 represents a newly-created layout that has not been saved to a file.

Figure 1-1. Layouts Menu - Handles and Filenames



You typically want to use the Tcl **set** command to assign a variable to a handle as described in “[Variables](#)” on page 22.

Layout Viewer Commands and Object Methods

The layout viewers support the use of commands and object methods for manipulating the layout viewer application, current layout, current view, and other state-dependent data.

The layout viewer commands and object methods are documented in “[Layout Viewer Batch Commands and Object Methods](#)”. They are organized into sections based on whether it is a command or object method, and the object type (cwbWorkBench, layout, overlay, peek, cwbLayoutView, and StringFeature) they are designed to operate on.

- **Commands** are designed to perform general manipulations of the layout viewer session, layouts, overlays, peek objects, and views of the current layout. These commands do not require a handle.
- **Object methods** are designed to manipulate a specific layout viewer session, layout, overlay, peek, or view object. Object methods require a handle that identifies the specific object on which the command should operate. Refer to “[Layout Viewer Objects and Handles](#)” on page 24 for more information about objects and handles.

Within the reference material for this document, object method names are written using the format “\$<variable name> <operation>”, representing the command you use if you assign the instance name to one of the following variable names:

Object Class	Variable Name
cwbWorkBench	cwb
Layout	L
Overlay	O
cwbLayoutView	V
StringFeature	str

Most Database Objects are not Tcl Objects

All Tcl objects have handles. The Calibre layout viewer classes of Tcl objects include cwbWorkBench, Layout, Peek, Overlay, StringFeature, and cwbLayoutView. Many layout database objects are not Tcl objects. Within memory, these layout database objects 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 object methods 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 [Layout Object Methods](#) to operate on database objects, such as cells, polygons, references, and layers.

Layout Viewer Application Extension Files

You can write Tcl-based scripts in the supported application extension files to create customized macros or modify the behavior of the Calibre layout viewers.

The supported application extension files for the Calibre layout viewers include:

- *wbinit.tcl* — This file is read after the Tcl package is initialized. When you run Calibre DESIGNrev in shell mode (using the `-shell` or `-a` option), Tcl is loaded without the Tk package. If the *wbinit.tcl* file contains any Tk commands, they cannot be executed, so the application will exit with an error. Refer to “[wbinit.tcl File Format](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for more information on using this file.
- *wbposttkinit.tcl* — This file can contain both Tcl and Tk commands. It is read after both the Tcl and Tk packages are initialized. Refer to “[wbposttkinit.tcl File Format](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for more information on using this file.

Questions and Answers

This section provides answers to some frequently asked questions about using Tcl.

Q: Where do I get layout handles?

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

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 can choose the variable name. Handles are system generated by the layout viewer, so you have no control over the handle assigned to an object. 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 saves 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 shown in [Table 1-2](#), but with caution. Preferably, capture handles as the application returns them and assign them to a variable. This helps guarantee you are pointing to the correct objects. Using built-in handles should be avoided in batch mode.

Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

Table 1-3. Syntax Conventions

Convention	Description
Bold	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.
UPPerCase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.

Table 1-3. Syntax Conventions (cont.)


Convention	Description
' '	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.
Example: DEVICE { <i>element_name</i> ['('model_name')] } <i>device_layer</i> { <i>pin_layer</i> ['('pin_name')] ...} ['<'auxiliary_layer'>' ...] ['('swap_list')' ...] [BY NET BY SHAPE]	

Chapter 2

Getting Started

Calibre DESIGNrev provides multiple options for running the tool, depending on whether you want to work in the layout viewer GUI, execute a Tcl script, or simply execute a command. These options include: Interactive GUI, Interactive shell, Batch, Batch GUI, and Command.

Note

 Specifying the **calibredrv** command without any options invokes Calibre DESIGNrev in interactive GUI mode. Starting the application as a background process will cause the Tcl shell to lock up.

Issuing Tcl Commands	30
Tcl and Shared Libraries	35
Explorations in Invocation Options	36
Explorations In Writing Procedures	43
Questions and Answers	52

Issuing Tcl Commands

There are multiple options available for invoking Calibre DESIGNrev and issuing Tcl commands.

Table 2-1 provides a summary of the different run options, including whether or not the option uses Tk.

Table 2-1. Summary of Options

Options	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.

Interactive GUI	30
Interactive Shell	32
Batch	33
Batch GUI	34
Command	34

Interactive GUI

The interactive GUI provides access to both the layout viewer and the terminal (or shell) window. You invoke the interactive GUI whenever you issue a layout viewer executable without specifying a script or the -a or -shell arguments.

For example:

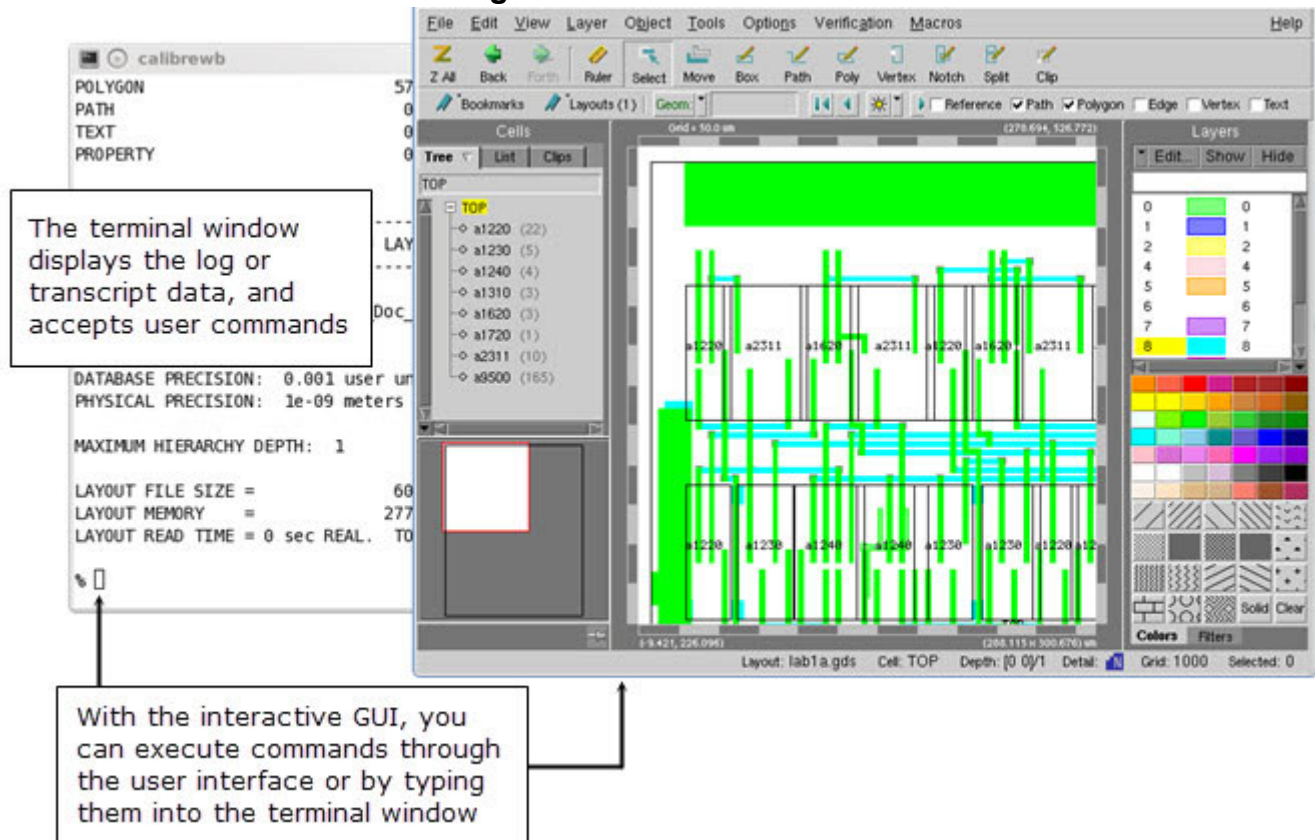
- Calibre DESIGNrev
`calibredrv lab1.gds`
- Calibre LITHOview
`calibrelv lab1.gds`
- Calibre MDPview
`calibremdp lab1.gds`

- Calibre WORKbench

```
calibrewb lab1.gds
```

In the layout viewer, you issue commands using the mouse, menus, and dialog boxes. In the terminal window, you can enter commands at the Tcl prompt.

Figure 2-1. Interactive GUI



Calibre RVE is a graphical debug program that interfaces with most IC layout tools. To invoke Calibre RVE with Calibre DESIGNrev, you can specify the `-rve` option when invoking Calibre DESIGNrev and optionally specify an RVE file and arguments. For example:

```
% calibredrv lab1.gds -rve lab1.results
```

For more information on Calibre RVE, refer to the [Calibre RVE User's Manual](#).

For information on the interactive GUI invocation arguments, refer to the [calibredrv](#) command reference page in the *Calibre DESIGNrev Layout Viewer User's Manual*.

Interactive Shell

When you invoke the interactive shell, you have access only to the shell 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 an interactive shell by entering the layout viewer executable with the `-shell` argument. For example:

- Calibre DESIGNrev

```
calibredrv -shell
```

- Calibre LITHOview

```
calibrelv -shell
```

- Calibre MDPview

```
calibremdp -shell
```

- Calibre WORKbench

```
calibrewb -shell
```

Upon entering one of the above commands, the shell window prompt changes to the Tcl “%” prompt and you can then execute commands by typing them into the shell window at the Tcl prompt. For example:

```
% calibredrv -shell

// Calibre DESIGNrev <version>
//
// Copyright Siemens 1996-2021
// 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.
//

% set layout [layout create lab2.gds]
```

The tool writes transcript information to the shell window as it does with the interactive GUI option.


```
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/drvtlab/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 tool also outputs any data that is returned by a command entered directly in the shell window. For example:

```
% layout peek lab1a.gds -cells
TOP a9500 a1310 a1240 a1230 a1720 a2311 a1220 a1620
```

For information on the interactive shell invocation arguments, refer to the [calibre drv -shell](#) command reference page in the *Calibre DESIGNrev Layout Viewer User's Manual*.

Batch

Batch refers to the processing of a script containing commands that are executed on one or more files. In the case of the Calibre layout viewers, a batch script contains Tcl commands and is typically executed on layout data.

To execute a layout viewer with the batch option, you specify the layout viewer executable and the name of the Tcl script. For example:

- Calibre DESIGNrev
calibredrv myscript.tcl
- Calibre LITHOview

```
calibrelv myscript.tcl
```

- Calibre MDPview

```
calibremdp myscript.tcl
```

- Calibre WORKbench

```
calibrewb myscript.tcl
```

When using this option, only the Tcl package is available to you so you cannot execute any Tk commands or any layout viewer commands that require a GUI. For information on the batch invocation arguments, refer to the [calibredrv <script.tcl>](#) command reference page in the *Calibre DESIGNrev Layout Viewer User's Manual*.

Batch GUI

With the batch GUI option, both the Tcl and Tk packages are available when performing batch processing.

To run a layout viewer with the batch GUI option, you specify the layout viewer executable and the name of the Tcl script. For example:

To execute a Tcl script using the batch GUI option, you specify the layout viewer executable, the name of the Tcl script, and the `-gui` argument. For example

- Calibre DESIGNrev

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

- Calibre LITHOview

```
calibrelv myscript.tcl -gui
```

- Calibre MDPview

```
calibremdp myscript.tcl -gui
```

- Calibre WORKbench

```
calibrewb myscript.tcl -gui
```

For information on the batch GUI invocation arguments, refer to the [calibredrv <script.tcl> -gui](#) command reference page in the *Calibre DESIGNrev Layout Viewer User's Manual*.

Command

You can use the (non-interactive) `command` option to evaluate a single Tcl command and then exit. This option does not execute any Tk commands or any of the layout viewer commands that require a GUI.

You invoke the command option by entering the layout viewer executable with the `-a` argument. If you execute a batch command that returns a data string, you can explicitly output the results to a file or the transcript using the Tcl **puts** command with a redirect. For example:

- Calibre DESIGNrev

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

- Calibre LITHOview

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

- Calibre MDPview

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

- Calibre WORKbench

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

For information on the command invocation arguments, refer to the [calibredrv -a <command>](#) command reference page in the *Calibre DESIGNrev Layout Viewer User's Manual*.

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 a Calibre layout viewer, to specify a path to the shared library.

Explorations in Invocation Options

Three different examples explore the differences you encounter when performing a single task using a layout viewer invocation option.

Example 1 - Listing the Cells in a Layout Using the Interactive Shell Option	36
Example 2 - Listing the Cells in an Open Layout Using the Interactive GUI Option..	38
Example 3 - Listing the Cells in a Layout Using Batch Mode	41

Example 1 - Listing the Cells in a Layout Using the Interactive Shell Option

In this example, you invoke Calibre DESIGNrev in shell mode and use Tcl to generate a list of the cells in a layout.

The topics introduced include:

- 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 object method.

Prerequisites

- A layout located in the directory from which you invoke the tool. This example uses a GDS file named *sample.gds*.

Procedure

1. Invoke Calibre DESIGNrev in shell mode:

```
$ calibredrv -shell
```

2. Enter the following command at the “%” prompt:

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

This line includes two commands:

- **set** — A basic Tcl command that creates a variable named “my_layout” and assigns the variable the value of the data returned by the layout create command.
- **layout create** — A Layout command (see “[Layout Commands](#)” on page 152) that reads the GDS file into the layout object it creates.

Example 1 - Listing the Cells in a Layout Using the Interactive Shell Option

The brackets instruct Tcl to replace the brackets and everything between with the results from executing the command inside the brackets (referred to as *command substitution*). In this case, the layout create command creates a new layout from sample.gds and assigns the new layout the system generated name “layout0” (also called a handle). The command “set my_layout [layout create sample.gds]” becomes “set my_layout layout0”.

When you enter the command, the layout viewer processes the command, displays the transcript information¹ generated when loading the layout, then prints the result of processing², in this case “layout0”.

When processing is complete, the tool displays the Tcl prompt (%), 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. Enter the following command at the “%” prompt:

```
% $my_layout cells
```

This command is actually the layout object method [\\$L cells](#). Layout object methods are a set of layout viewer commands that operate on an instance of an object (and only that

1. Whenever you run a Calibre layout viewer, the tool 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.

instance). In this example, you created an instance of a layout object in step 2, using the command `layout create sample.gds`, and assigned the handle to the variable name “`my_layout`”. This allows you to reference the layout object using the string “`$my_layout`”.

4. Exit the application.

```
% exit
```

Results

When the layout viewer evaluates the command it returns a list of the cells found in `sample.gds`. When processing is complete, it displays the “%”, indicating it is ready for you to issue a new command.

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

Refer to “[Layout Viewer Batch Commands and Object Methods](#)” on page 99 for information on the available layout viewer commands and object methods.

Related Topics

[layout create \(GDS or OASIS file\)](#)

[\\$L cells](#)

[Layout Viewer Objects and Handles](#)

Example 2 - Listing the Cells in an Open Layout Using the Interactive GUI Option

In this example, you work with a layout that is 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 are needed elsewhere.

The topics introduced include:

- The iTcl “find objects” command.
- Combining variable substitutions and command substitution.

In Example 1, you used the “`layout0 cells`” object method to list the cells in `layout0`. Because the name of the method is of the form *<instance name> <operation>*, you cannot use this method 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.

Prerequisites


- A sample GDS file named *sample.gds* exists in the directory from which you invoke the tool.

Procedure

1. Invoke Calibre DESIGNrev in interactive GUI mode:

```
$ calibredrv sample.gds
```

Note

 In Example 1, you used the layout create command to create a layout in the Calibre database. However, layout create did not load the layout into the GUI because it is not a Tk command. To load a layout file into the GUI, you must use the `$cwb viewedLayoutClbk` command.


2. Click in the terminal window to make it the active window. You may need to move, resize, or minimize the GUI to access the terminal window.
3. Press Enter to access the “%” prompt.
4. Enter the following command at the “%” prompt:

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

This line combines two commands:

- `set` — Creates the variable “cwb” and assigns it the result of the `find objects` command.
- `find objects` — Finds any object of class `cwbWorkBench`. Since you have only one Calibre DESIGNrev window open in this session, the command returns only one value, which is the system-generated name for this object, in this case “::3


Note

 This example assumes you only have one layout open. If you have more than one layout open, the `find objects` command 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.

3. The double colon “::” is an iTcl construct used to identify and manage namespaces. Unless you go on to advanced Tcl programming, you do not need to worry about iTcl namespaces. You can find more information on namespaces at: <http://incrtcl.sourceforge.net/itcl/namesp.html>

Note

 Even if you are running one of the Calibre layout viewer applications other than Calibre WORKbench, you need to create a Calibre WORKbench object. The name of the application window object used by all of the Calibre layout viewers is “cwbWorkBench” and some of the hard-coded variable names and handles begin with “cwb”.

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

5. Enter the following command at the “%” prompt:

```
% 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 currently 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 brackets and everything between them with the results of processing the command inside, resulting in:

```
set my_layout layout0
```

6. Enter the following command at the “%” prompt:

```
% $my_layout cells
```

Due to variable substitution, the line you type 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 and Example 2, it is just as easy to skip defining the variables and simply retype the values returned by various Tcl commands. However, in most practical situations, you do much more complex work, in which using variables becomes necessary.

7. Exit the application.

```
% exit
```

Related Topics

[\\$cwb viewedLayoutClbk](#)

[\\$L cells](#)

[\\$cwb cget -_layout](#)

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

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

The topics introduced include:

- Displaying results using the Tcl “puts” command.

To list the cells in a layout using batch mode, the logical thing to do is write a script containing only those commands you issued at the prompt when working in interactive shell mode.

However, 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.

Prerequisites

- A sample GDS file named *sample.gds* exists in the directory from which you invoke the tool.

Procedure

1. Use an ASCII editor to create the following Tcl script, adding the Tcl “puts” command to the last line.

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

The Tcl “puts” command 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. Save the file as *DisplayCellList.tcl*.
3. Invoke Calibre DESIGNrev in batch mode by passing in the name of the script:

```
$ calibredrv 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

%
```

Related Topics

[layout create \(GDS or OASIS file\)](#)

[\\$L cells](#)

Explorations In Writing Procedures

Two examples are used to explore the basics of writing procedures, using examples that display an entire layout hierarchy.

For these examples, assume the file *sample.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 43

Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy..... 47

Example 4 - Using a Simple Procedure to Write the Layout Hierarchy

In this example, you create a simple procedure to write the layout hierarchy. The Tcl commands used in this example include open, puts, set, proc, foreach, and read.

The topics introduced include:

- Writing results to a file.
- Creating a procedure.
- Using the foreach Tcl command to look through an array of objects.
- Displaying the contents of a file inside the Calibre DESIGNrev terminal window.

Prerequisites

- A sample GDS file named *sample.gds* exists in the directory from which you invoke Calibre DESIGNrev. The design, *sample.gds*, used in this procedure has three levels of hierarchy.

Procedure

1. Invoke Calibre DESIGNrev in GUI mode and open the *sample.gds* layout.

```
$ calibredrv sample.gds
```

2. In the terminal window, press Enter to access the Tcl “%” prompt.

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

```
% set fileID [open dump_hier.txt w]
<fileID>
% puts $fileID "This file displays the hierarchy for sample.gds"
```

In Interactive mode, Calibre 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.

In this step, the Tcl commands do the following:

- **set** — Assigns the “fileID” variable the results of the command in the brackets.
- **open** — Opens the *dump_hier.txt* file and returns a channel identifier, *<fileID>*, that is used to communicate with that file. The w (write) argument tells Tcl to open the file as write-only. Tcl creates the file if it does not exist.
- **puts** — Writes to the channel assigned to \$fileID. This command takes two arguments, the output channel (where to print the information) and the data (what to print). Using the channel identifier (\$fileID) as the first argument instructs the command to write the data to the file rather than the terminal window (stdout). The second argument is a string enclosed in quotes that specifies the data to write to the file. Since the string contains spaces, you must enclose it in quotes in order for the command to treat the string as a single object.

4. Get the layout name:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout0
```

5. 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 that the brackets are for command substitution and the \$ for variable substitution. 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 \$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 want to print special characters such as “[”, “]” or “\$”, you can do so by preceding them with a backslash “\”.

6. Next, write a procedure to print out the children of a given cell. You 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 take the following form:

proc <name> <arguments> <body>

- **proc** — Specifies this is a Tcl procedure.
 - *name* — Specifies a name for the procedure. In this example, the name of the procedure is “list_children”.
 - *arguments* — Specifies the formal arguments to the procedure. The arguments are always enclosed in braces ({ }). In this example, there are three arguments: L, F, and C. Within the body of the procedure, these arguments are used as variables.
 - *body* — Contains the body of the procedure (enclosed in braces) that the Tcl interpreter executes, even if it spans multiple lines and contains multiple commands. In this example, 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.
7. 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 $L $fileID $stop
```

8. 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 $stop] {
    list_children $L $fileID $cell
}
```

The foreach command creates a *loop variable* named “cell” and assigns it the values resulting from processing the command in brackets. It cycles through the values one at a time, evaluating the commands between the braces ({ }). Since the second level of the hierarchy in *sample.gds* contains two cells, NAND2A and NOR, this is equivalent to the following two statements:

```
list_children $L $fileID NAND2A
list_children $L $fileID NOR
```

9. Optionally, save the pathname of the invoking tool:

```
% puts $fileID [info nameofexecutable]
/clbr_latest/ic/ixl/Mgc_home/pkgs/icwb/pvt/calibrewb
```

10. Save and close the file:

```
$ close $fileID
```

11. Reopen the file to view its contents:

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

In this step, the Tcl commands do the following:

- **open** — Opens the *dump_hier.txt* file and returns a channel identifier. The file was originally opened using the w (write-only) argument in step 3. In this step, you specify the r (read) argument in order to open and read the data in the file.
- **read** — Returns the information read by the open command to the application. This command requires at least one argument, the input channel (where to find the information). In either interactive mode, the application prints the return values for all commands to the shell.

12. Close the file:

```
% close $fileID
```

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

13. Open the resulting *dump_hier.txt* file to review the results.

Results

The results are written to the shell and to the file named *dump_hier.txt*. If you are working with a layout that has additional levels of hierarchy, you can add another foreach statement that calls the procedure to list the next level in the hierarchy. For example, if your layout contains four levels of hierarchy, you would replace the foreach statement shown in step 8 with the following:

```
% foreach cell [$L children $stop] {  
    foreach child [$L children $cell] {  
        list_children $L $fileID $child  
    }  
}
```

Working in interactive GUI mode, you can continue to add these looping statements as needed to cover the walk through the entire hierarchy. This is an inefficient way to solve the problem, because it depends on you being able to anticipate how many levels of hierarchy are in the layout. For a more efficient approach, refer to “[Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy](#)”.

Examples

Here's a transcript of writing the layout hierarchy:

```
$ calibredrv sample.gds

% set fileID [open dump_hier.txt w]
<fileID>
% puts $fileID "This file displays the hierarchy for sample.gds"
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout0
% puts $fileID "Topcell: [set top [$L topcell]]"
% proc list_children {L F C} {
    puts $F "Children of $C: [$L children $C]"
}
% list_children $L $fileID $top
% foreach cell [$L children $top] {
    list_children $L $fileID $cell
}
% close $fileID
% set fileID [open dump_hier.txt r]
<fileID>
% read $fileID
This file displays the hierarchy for sample.gds
Topcell: TOP
Children of TOP: NAND2A NOR
Children of NAND2A: NAND
Children of NOR:

% close $fileID
```

Related Topics

[\\$cwb cget -_layout](#)

[\\$L children](#)

[\\$L topcell](#)

Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy

In this example, you 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.

Topics introduced:

- Comments.
- Procedure arguments with default values.

- Recursive procedures.
- Local versus global variables.

Prerequisites

- A sample GDS file named *std_.gds* exists in the directory from which you invoke Calibre DESIGNrev.

Procedure

1. Create a Tcl script named *dump_hier.tcl*.
2. Add a comment block to introduce the procedure and define its arguments:

```
#####
# 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
#####
```

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.

3. Add the following Tcl procedure to traverse the cell hierarchy and print each level:

```
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 is “”, 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 `append` command adds additional characters to an existing variable (in this case, `$Indent`). Here it shows 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 calls 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.

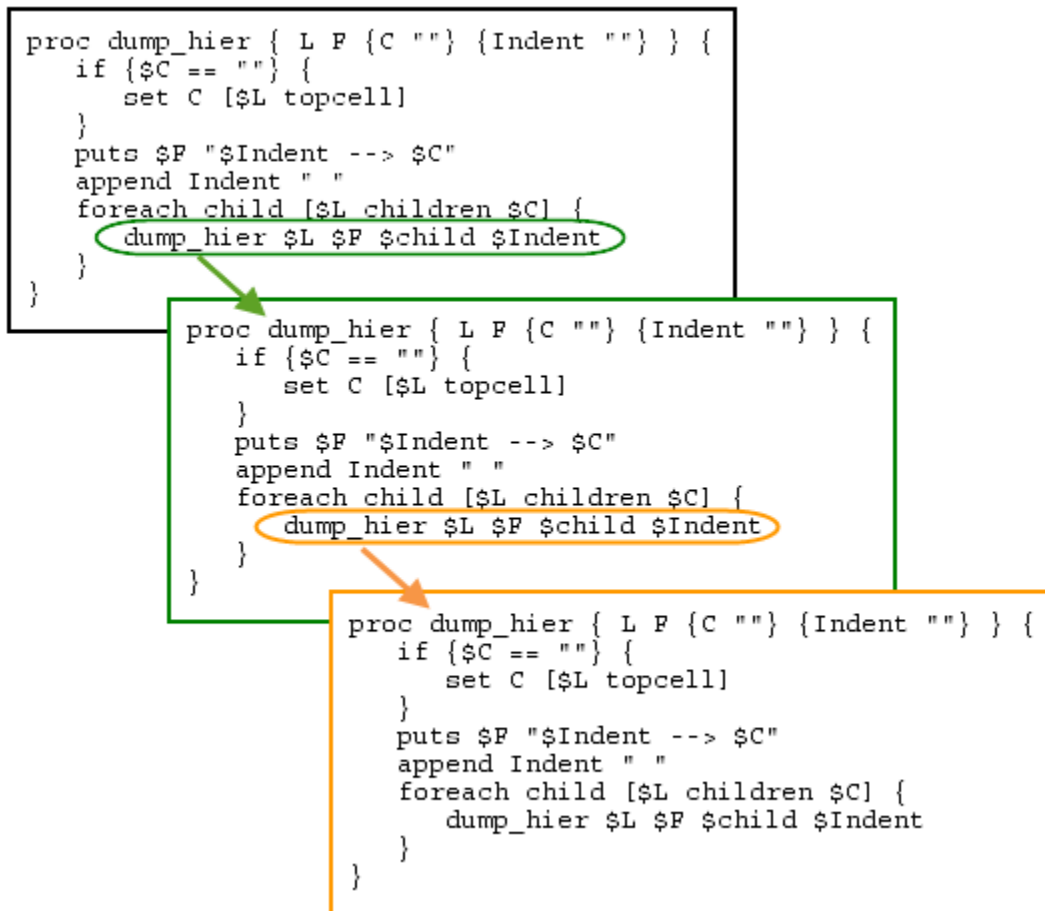
Recursive procedures are possible because Tcl supports the concept of variable scope. Scope defines where a variable has meaning. Scope can be either local or global:

- *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. 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 a procedure within a procedure, within a procedure.

All the variables in your recursive procedure are local. This means the variable `F` in the lowest level procedure, shown outlined with an orange dashed line in [Figure 2-2](#), is a completely different variable from the variable `F` in the middle procedure, shown outlined with a green solid line.

Figure 2-2. Graphical Representation of a Recursive Procedure



4. Add the following code snippet to *dump_hier.tcl* to create or open the output file, open the input layout file, write the hierarchy to the output file, and close the output file:

```

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

```

5. Invoke Calibre DESIGNrev:

```
$ calibredrv dump_hier.tcl
```

6. View the file you have created:

```
$ more dump_hier.txt
```

Results

You have a script, *dump_hier.tcl*, that writes the layout hierarchy for an input layout file to a text file, *dump_hier.txt*. The completed script should resemble the following:

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

Related Topics

[\\$L children](#)

[\\$L topcell](#)

[layout create Commands](#)

Questions and Answers

This section provides answers to some frequently asked questions about using the layout viewer tools and Tcl.

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

A: All of the exercises in this manual do apply to you. Calibre DESIGNrev provides a subset of the capabilities provided by the Calibre WORKbench tool. This means any scripts that run in Calibre DESIGNrev will also run in Calibre 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 Calibre WORKbench object. Should I be creating a Calibre DESIGNrev object?

A: There is no Calibre DESIGNrev object. The application window object is called *cwbWorkBench*, regardless of which Calibre layout viewer you are running. The *cwbWorkBench* object is configured differently to create the different applications. Notice that some of the hard-coded variable names and handles begin with “cwb”.

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

A: The backslash “\” at the end of a line indicates the command does not fit onto one line and the command continues on the next line. When you use the backslash as a command continuation character, it must not be followed by any other characters. Even a space after it creates an error.

Chapter 3

Writing Layout Viewer Scripts

Tcl scripts are stand-alone programs you write using the Tcl language, plus any extensions you have available to you. For the Calibre layout viewer tools, Tcl scripts are written using the layout viewer Tcl extensions and batch commands.

The objects and commands you are most likely to use in scripts are the [cwbWorkBench Object Methods](#) and the [Layout Object Methods](#).

Executing Tcl Scripts.....	53
Scripts for Layout Generation and Assembly.....	54
Script 1: Writing a Script to Generate a New Layout.....	54
Script 2: Setting up a Script to Accept Passed In Arguments.....	56
Script 3: Debugging and Adding Error Handling.....	59
Script 4: Moving a Cell's Origin	64
Script 5: Changing Cell Names in a GDS File	66

Executing Tcl Scripts

You can execute Tcl scripts in batch or interactive mode.

Procedure

Use one of the following methods to set to execute a Tcl script:

If you want to...	Do the following:
Execute a Tcl script in batch mode, by including the pathname for the script when invoking the layout viewer	<pre>calibredrv my_script.tcl calibredrv my_script.tcl -gui</pre>
Execute a Tcl script in interactive mode, by sourcing the script from the command line	<pre>% source my_script.tcl</pre>

Scripts for Layout Generation and Assembly

You can create Tcl scripts to automate the process of modifying or combining database objects. *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 or changing database units.

Scripted layout assembly involves taking existing cells or layout files and combining them into a new layout. Scripted layout assembly is a type of very high level layout editing – it never gets down to the polygon level. Assembly can 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	54
Script 2: Setting up a Script to Accept Passed In Arguments	56
Script 3: Debugging and Adding Error Handling	59
Script 4: Moving a Cell’s Origin	64
Script 5: Changing Cell Names in a GDS File.....	66

Script 1: Writing a Script to Generate a New Layout

You can create a Tcl script for generating a new layout using batch commands. Topics covered in this procedure include:

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

Procedure

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

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

This line combines two commands:

- **set** — This Tcl command creates the variable L and assigns it the return value from the layout create command.
- **layout create** — This layout command creates a new layout and 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 add data to this new layout.

2. Create a topcell within this layout:

```
$L create cell TOP
```

The \$L create cell command creates the cell TOP in the new layout. It is a topcell 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 2.7 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 temp.gds
```

Results

You should now have a Tcl script named *new_layout.tcl* that creates a GDS file named *temp.gds* containing one cell, two layers, and two polygons.

```
set L [layout create]
$L create cell TOP
$L create layer 1
$L create layer 2.7
$L create polygon TOP 1 0 0 100 100
$L create polygon TOP 2.7 0 0 0 40 30 40 30 80 45 80 45 25 15 25 15 0
$L gdsout temp.gds
```

Related Topics

[layout create](#)

`$L create cell`

`$L create layer`

`$L create polygon`

`$L gdsout`

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

You can use a Tcl script to generate a new layout and accept command line arguments that specify the name of the topcell, a layer number, the coordinates of a polygon, and the name to use for saving the new GDS layout.

The Tcl commands and variables covered in this procedure include:

- **index** — Extracts one or more elements from a Tcl list. The syntax for this command is:

```
index list index
```

The first argument is the name of the list and the second argument is the position of an element to return. The first element in a list has an index of 0 and the last element in a list of length *<N>* has an index of *<N>-1*.

- **argv** — A global Tcl variable that is a list of all command line argument values.
- **argc** — A global Tcl variable that contains the number of command line arguments in argv.
- **lrange** — Returns one or more adjacent elements from a list. The syntax for this command is:

```
lrange list first last
```

The first argument is the name of the list, while *first* and *last* specify the range of elements to return.

Prerequisites

- The *new_layout.tcl* script (refer to “[Script 1: Writing a Script to Generate a New Layout](#)” on page 54).

Procedure

1. Open the *new_layout.tcl* Tcl script.
2. Modify the following line of the script to accept input that specifies the name of the topcell you want to create:

- **Change:**

```
$L create cell TOP
```


- **To:**

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

These lines contain the following commands:

- **lindex \$argv 0** — Returns the first argument (cell name) in the arguments list.
- **set my_cell [lindex \$argv 0]** — Stores the results of the lindex command in the variable my_cell.
- **\$L create cell \$my_cell** — Creates a cell using the results from processing the lindex command and assigning the name of the my_cell variable to the new cell.

3. Modify the following lines of the script to accept input that specifies the numbers of the layers you want to create:

- **Change:**

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

- **To:**

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

This line contains the following commands:

- **lindex \$argv 1** — Returns the second argument (layer number) in the arguments list.
- **set layer_num [lindex \$argv 1]** — Stores the results of the lindex command in the variable layer_num.
- **\$L create layer [set layer_num [lindex \$argv 1]]** — Creates a layer using the results from processing the lindex command, which is assigned to the variable layer_num. 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.

4. 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 Tcl 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 Tcl 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].

5. 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 Tcl **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 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.

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

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

The Tcl **lindex** command recognizes the end of the string as representing the last index in a list.

7. Save the script.

8. Run the script:

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

9. View the new layout in Calibre DESIGNrev:

```
calibredrv new_file.gds
```

Results

You have now modified the *new_layout.tcl* Tcl script to define arguments and lists, and extract polygon coordinates.

```
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]
```

When executing the script, in addition to specifying the name of the script, you must specify the following information on the command line:

```
calibredrv new_layout.tcl <cellName> <layer> <polygon_coordinates>
<output_file>
```

Related Topics

[layout create](#)

[\\$L create cell](#)

[\\$L gdsout](#)

[\\$L create polygon](#)

[\\$L create layer](#)

Script 3: Debugging and Adding Error Handling

You can use a Tcl script to introduce errors and then find and fix them for a more realistic programming experience. You also anticipate possible user errors and build in error handling for those situations.

Topics covered in this procedure include:

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

Prerequisites

- The *new_layout.tcl* script (refer to “[Script 2: Setting up a Script to Accept Passed In Arguments](#)” on page 56).

Procedure

1. Make the following changes to *new_layout.tcl*:

- **Change:**

```
set L [layout create]
$L create cell $my_cell
eval $L create polygon
```

- **To:**

```
set L [layuot create];# <-- Note intentional misspelling!
$Ll create cell $my_cell
$L create polygon
```

Your script should appear as follows:

```
set L [layuot create];# <-- Note intentional misspelling!
set my_cell [lindex $argv 0]
$Ll 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. Save the script as *error.tcl*.
3. Run the script.

```
calibredrv error.tcl
```

You will get an error message similar to the following:

```
invalid command name "layuot" <-- Type of error
while executing
"::_unknown_ layuot create"
("uplevel" body line 1) <-- Line number
invoked from within
"uplevel 1 [linsert $args 0 ::_unknown_]"
(procedure "::_unknown" line 6)
invoked from within
"layuot create"
invoked from within
"set L [layuot create]"
(file "error.tcl" line 1)
```

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

4. Edit *error.tcl* to correct the spelling mistake in the first line (change “layuot” to “layout”). Save the file and run the script again.

```
calibredrv error.tcl
```

You will get error messages similar to the following:

```
can't read "L1": no such variable
while executing
"$L1 create cell $my_cell"
(file "error.tcl" line 3)
```

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

5. Edit *error.tcl* and, in the second line, change \$L1 to \$L. Save the file and run the script again.

```
calibredrv error.tcl
```

You will get error messages similar to the following:

```
Error: Cell name "" is empty or contains either SPACE character or
non-printable ASCII character.
while executing
"$L create cell $my_cell"
(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. In this case, the problem is that “[set layer [lindex \$argv 1]]” has no meaning because no arguments were passed to argv.

6. Edit *error.tcl* and add the following code to the beginning of the script to define arguments for argv:

```
#####
#
# 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) --> filename
#
#####

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

This code checks to see if you passed in enough arguments. If not, the script lets you know exactly what arguments to supply and exits the program.

7. Save the file and run the script with no arguments.

```
calibredrv error.tcl
```

You will get error messages similar to the following:

```
Wrong number of arguments
<cell name> <layer number> <coords> <filename>
```

For the remaining steps, you must supply the seven arguments (identified in the comment text in step 6) in order to run and debug your script.

8. Run the script again, this time specifying the minimum required arguments:

```
% calibredrv error.tcl top 4 0 0 100 100 test.gds
```

You will get error messages similar to the following:

```
Usage: layout0 create polygon: cell L_D [-prop attr string [G|U|1]]*
x1 y1 ... xn yn
    while executing
"$L create polygon $my_cell $layer1 $coords"
    (file "error_step6.tcl" line 24)
```

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 or command substitution does not execute, you should try using the Tcl **eval** command before you assume the data is wrong.

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

9. Edit *error.tcl* and add the **eval** command that you removed in step 1 back in front of \$L create polygon.
10. Save the script and run it again.

The script should run without errors.

11. Run the script again, this time passing in an odd number of coordinate values. For example:

```
% calibredrv error.tcl top 4 0 0 100 100 70 test.gds
```

You will get error messages similar to the following:

```
Error: layout0 create polygon top: found uneven number of coordinate
values.
    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_step9.tcl" line 24)
```

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.

12. Edit *error.tcl* and add the following code just below the line that extracts the coordinates (set coords) from the arguments list:

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

Here is an explanation of the Tcl commands and operator used in this code:

- **if** — A Tcl command that evaluates the `expr` command as an expression. The result is a boolean, where 0 is false and any other value is true.
- **llength** — A Tcl command used to return the number of elements in a list.
- **expr** — A Tcl command that evaluates the expression “`$len % 2`”, treating a value as a number and not a string.
- **%** — A Tcl arithmetic operator that checks if there is an even number of coordinates. This operator returns the remainder after dividing the first value (`$len`) by the second (2).

Thus, this code prints an error message and exits if the remainder of `$len` divided by 2 does not equal 0.

13. Run the script to verify it works.
14. Add comments to make the script more readable.

Try to think of other possible error conditions and add conditional statements to validate arguments and supply usage commentary. Things you might consider doing to get more out of this exercise include using the following Tcl commands:

- Use **catch** with [\\$L create layer](#). If there is an error, warn the user the second argument must be a number.
- Use **string is** to check that coordinates are actually numbers.
- Use **catch** with [\\$L gdsout](#) in case the program cannot open the file for writing. Note that you may want to use [\\$L oasisout](#) instead of `$L gdsout` to output the results.

Note



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

15. Save the script to *no_error.tcl*.

Results

Here is the completed *no_error.tcl* script:

```
#####
# 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) --> filename
#####
if {$argc < 7} {
    puts "wrong number of arguments"
    puts "<cell name> <layer number> <coords> <filename>"
    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]
```

Related Topics

[layout create](#)

[\\$L create cell](#)

[\\$L gdsout](#)


[\\$L create polygon](#)

[\\$L create layer](#)

Script 4: Moving a Cell's Origin

You can use batch commands to move the origin of the topcell in an existing layout.

Note

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

Topics covered in this procedure include:

- Creating cell references. Cell references do not contain geometric data.
- Moving the origin of a cell.
- Setting constants for convenience.

Prerequisites

- An existing layout named *mylayout.gds*.

Procedure

1. Create a Tcl script named *move_cell_origin.tcl* and add the following lines of code:

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

These lines do the following:

- Sets a variable for the name of the original layout file.
- Sets a variable for the resulting layout.

2. Set variables for the X and Y deltas in user units.

```
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) and assign it to the variable “mydbunits”. A database unit (dbu) is the unit measure of length for the input database.

```
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
```

9. Save the script.

10. Run the script:

```
calibredrv move_cell_origin.tcl
```

Results

Here is the completed *move_cell_origin.tcl* script:

```
set mylayout mylayout.gds
set myoutfile myoutfile.gds
set mydeltax 12.0
set mydeltay 15.0
set L [layout create $mylayout -dt_expand]
set mytopcell [$L topcell]
$L cellname $mytopcell myoldtopcell
set mydbunits [$L units user]
$L create cell $mytopcell
$L create ref $mytopcell myoldtopcell \
    [expr $mydeltax / $mydbunits] \
    [expr $mydeltay / $mydbunits] 0 0 1
$L expand cell myoldtopcell
$L gdsout $myoutfile $mytopcell
```

Related Topics

[layout create](#)

[\\$L expand cell](#)

[\\$L create cell](#)

[\\$L create ref](#)

[\\$L gdsout](#)

[\\$L modify origin](#)

Script 5: Changing Cell Names in a GDS File

You can create a Tcl script that takes an existing GDS file as input, renames the cells using the specified prefix, and outputs the results to the specified GDS file.

Prerequisites

- An existing layout.

Procedure

1. Create a Tcl script named *change_cell_names.tcl*.
2. Add the following lines to set the variable names for the infile, outfile, prefix, and cell_list_file (list of cells to exclude from renaming).

```
set infile [lindex $argv 0]
set outfile [lindex $argv 1]
set prefix [lindex $argv 2]
set cell_list_file [lindex $argv 3]
```

3. Create a check for the inputs to the script:

```
if { [info exists infile] != 1 || [info exists outfile] != 1 || \
    [info exists pname] != 1 } {
    puts stderr "Usage: calibredrv $argv0 infile outfile \
    prefix [cell_list]"
    exit 1
}
```

4. Create a check for the cell list file:

```
if { $cell_list_file != "" } {
    if {[catch {set cellfile [open ${cell_list_file} r]}]} {
        puts stderr "Error: Cannot open exclude cell list file for \
        reading"
        exit 1
    }
}
```

5. Set a variable for the cells to be excluded from renaming and assign the contents of the cell list file to the variable:

```
set exclude_cells [read $cellfile]
close $cellfile
} else {
    set exclude_cells ""
}
```

6. Open the input GDS file and return a list of cells in the layout.

```
# open the library gds
set L [layout create $infile -dt_expand]
# get cell cells
set C [$L cells]
```

7. Rename the cells except for those that are excluded:

```

set i 1
set eclist [join $exclude_cells " "]
foreach scell $C {
    set ex 0
    foreach ec $eclist {
        if { [regexp $ec $scell] } {
            set ex 1
            continue
        }
    }
    if { $ex == 0 } {
        $L cellname $scell ${pname}_${i}
        incr i
    } else {
        puts "Note: $scell not changed"
    }
}

```

8. Output the GDS file with the renamed cells.

```

#save
$L gdsout $outfile

```

9. Save the script.

10. If you want to exclude cells from being renamed, create a file named *cell_list* and specify the cells to be excluded.

11. Execute the script, specifying the name of the input file, output file, and the prefix to use when renaming the cells. If you created the *cell_list* file, specify the name of that file.

```

calibredrv -s change_cell_names.tcl in_file.gds out_file.gds
cell_prefix [cell_list]

```

12. Open the new layout in Calibre DESIGNrev to view the renamed cells:

```

calibredrv out_file.gds

```

Results

You should now have a Tcl script named *change_cell_names.tcl* that creates a new layout containing the renamed cells from the input file.

```

set infile [lindex $argv 0]
set outfile [lindex $argv 1]
set pname [lindex $argv 2]
set cell_list_file [lindex $argv 3]

if { [info exists infile] != 1 || [info exists outfile] != 1 || \
    [info exists pname] != 1 } {
    puts stderr "Usage: calibredrv $argv0 infile outfile \
    prefix [cell_list]"
    exit 1
}

```

```

if { $cell_list_file != "" } {
    if {[catch {set cellfile [open ${cell_list_file} r]}]} {
        puts stderr "Error: Cannot open exclude cell list file for \
        reading"
        exit 1
    }
    set exclude_cells [read $cellfile]
    close $cellfile
} else {
    set exclude_cells ""
}

# open the library gds
set L [layout create $infile -dt_expand]
# get cell cells
set C [$L cells]

# rename
set i 1
set eclist [join $exclude_cells " "]
foreach scell $C {
    set ex 0
    foreach ec $eclist {
        if { [regexp $ec $scell] } {
            set ex 1
            continue
        }
    }
    if { $ex == 0 } {
        $L cellname $scell ${pname}_${i}
        incr i
    } else {
        puts "Note: $scell not changed"
    }
}

#save
$L gdsout $outfile

```

Related Topics

[layout create Commands](#)

[\\$L gdsout](#)

[\\$L cells](#)

Chapter 4

Writing Macros

Macros are application extensions you write to add new functionality to a Calibre layout viewer application, such as Calibre DESIGNrev. You can then make the extensions accessible from the **Macros** menu.

About Writing Macros	72
Simple Macro Example	75
Explorations in Creating Macros	77
Macro Examples to Study and Learn	92
Questions and Answers About Macros	97

About Writing Macros


One of the many uses for macros is to provide access to the many Calibre batch tools (such as DRC, LVS, and RET), 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. Writing and accessing macros typically involves writing the command procedure, writing the GUI plug-in, registering the macro, and loading the macro.

The default **Macros** menu in Calibre DESIGNrev contains some Calibre layout viewer-supplied macros. These macros provide useful functionality and also serve as examples of the sorts of functionality you can add to the application.

Use the Macros commands described in “[Macros Commands](#)” on page 476 to create, delete, and manipulate items in the **Macros** menu.

Note



User-defined macro extensions involve working with the Calibre layout viewer extensions to Tcl. The proprietary object methods you use when creating macros are:

- [cwbWorkBench Object Methods](#) — The main layout editor widget.
- [cwbLayoutView Object Methods](#) — The view widget for the layout.
- [Layout Object Methods](#) — The Tcl object containing the layout data.

Writing Command Procedures.....	72
Writing GUI Plug-ins	73
Register the Macro.....	74
Load the Macro	74

Writing Command Procedures

You can write a 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.

When you learn to write macros, it is easiest to write the command procedure before writing the GUI plug-in. Writing the procedure first ensures you know what information to 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 the command procedure, 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.

The code you write to create a macro can be located in the *wbinit.tcl* file or a separate file.

- In the *wbinit.tcl* file — If you define the macro in the *wbinit.tcl* file, embed it within a conditional statement instructing the tool to ignore the macro when the Tk package is not loaded, as described in “[Tcl Versus Tk and What the Distinction Means to You](#)” on page 17. The *wbinit.tcl* file is discussed in the section “[wbinit.tcl File Format](#)” in the *Calibre® DESIGNrev™ Layout Viewer User’s Manual*.
- In a separate file — If you define the macro in a separate file, you need to load it into the application. Refer to “[Load the Macro](#)” on page 74 for information on loading a macro.

Writing GUI Plug-ins

The GUI plug-in defines how the macro interacts with the graphical user interface. Typically, the plug-in detects state-dependent information or queries for user input and then invokes the macro command procedure. It can also display data returned by the macro command procedure.

When you choose a macro from 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:

- Obtains the data.

You can obtain information to pass to the command procedure using the handle for the application main widget or by querying the user for information.

When writing a macro, the `cwbWorkBench` handle is considered “known” because the application passes it to the GUI plug-in procedure whenever you invoke a macro. You can obtain two types of information from the application:

- Database information, such as cells, layers, and contents of cells.
- State-dependent information, such as “current” objects, selections, cursor location, and viewer depth.

The examples in “[Getting Started](#)” on page 29 show how you can obtain database information (the handle for a layout, the cells, and the hierarchical relationship between the cells) from the application.

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 using the Tcl **gets** command 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.

- Invokes the command procedure.
- Optionally, displays a message to the user or updates the GUI to display the new data.

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 [Scwb updateDisplay](#) for a description of the `cwbWorkBench` command for updating the GUI.

Register the Macro

You register the macro with the tool using 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 registered, the macro name can be placed on the layout viewer `Macros` menu as a menu item.

Load the Macro

Once the macro is registered, you can issue the `Macros menu` command to refresh the `Macros` menu so the new command is available to you.

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

- Adding all relevant macro code to the *wbinit.tcl* file.
- Using the source command in the *wbinit.tcl* 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 17.

You can also load the new macro using either of the following methods:

- Call 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 you.
- Source the file containing the macro code from the shell window, then issue the Macros menu command.

Related Topics

[cwbWorkBench Object Methods](#)

[Macros menu](#)

[Macros create](#)

Simple Macro Example

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

Procedure

1. Create a Tcl script containing only a proc named *return_info*, which runs the commands you require:

```
proc return_info { } {  
    tk_messageBox -message "Procedure return_info was called." \  
    -type ok  
    ...  
}
```

Note



To write a more complicated macro, it is easiest to first write the command script, and then test the script before completing this step.

2. Add a second proc, `return_info_plug_in`, which invokes the command procedure from the GUI plug-in:

```
proc return_info_plug_in { wbHandle window} {  
    return_info  
}
```

3. Register the original Tcl procedure as a macro:

```
if {[info exists tk_version]} {  
    macros create "my macro" return_info return_info_plug_in  
}
```

The Macros Tcl command is only available when running the layout viewer's GUI, and not when running in shell mode. Therefore the call must be wrapped in an if block to avoid errors when the Tcl script is sourced.

4. Source the macro from your `$HOME/wbinit.tcl` file, and then invoke the layout viewer:

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

Results

The macro is available in the Macros menu of the layout viewer.

Explorations in Creating Macros

Examples are used to explore the process of creating a simple macro that writes the hierarchy of the current layout to a file the user specifies.

This macro builds on the examples provided in “[Explorations In Writing Procedures](#),” which taught you how to write a script to output 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.

Transforming a Simple Script into a Command Procedure	77
Exploring Ways to Obtain User-Supplied Data	80
Writing a GUI Plug-in Procedure	82
Displaying a Message to the User	83
Adding the Macro to the Menu	85
Building in Error Handling	86
Setting Up Your Macro to Load Automatically	89
Sample Macro Code.....	89

Transforming a Simple Script into a Command Procedure

This procedure describes how to turn a simple script into a command procedure. You learn about the requirements for a command procedure and how to source a Tcl script from the shell window.

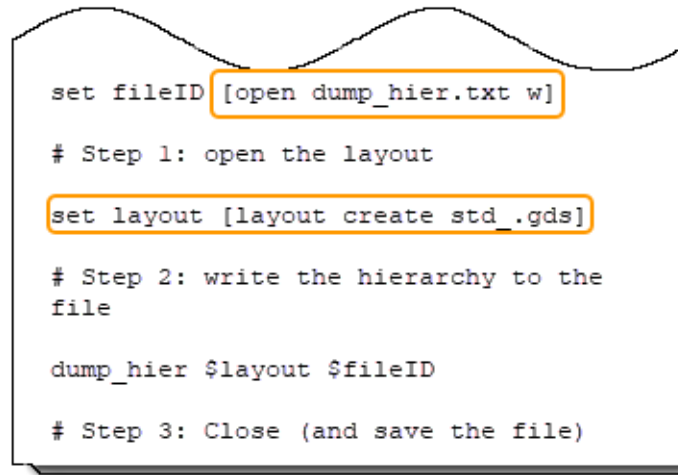
The command procedure performs all the data processing for the macro. After you have written and tested it, you write the GUI plug-in that calls this procedure.

Prerequisites

- The *dump_hier.tcl* file. Refer to “[Example 5 - Using a Recursive Procedure to Write a Layout Hierarchy](#)” on page 47 for information on creating this file.

Procedure

1. Open the *dump_hier.tcl* file and look for ways to make it more flexible:



```
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)
```

Some possible ways to make it more flexible include:

- Name of file to open — For maximum flexibility, change the hard-coded filename into a variable. Later, you write the GUI plug-in to get the full pathname for the file from the user.
- 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 of the procedure 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 serves as the command procedure.

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   = Calibre 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
}

```

Note

 The remaining steps test the functioning of the command procedure.

5. Invoke Calibre DESIGNrev in interactive GUI mode.
6. Load 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:

```
cat 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
```

Related Topics

[Writing Command Procedures](#)

[\\$cwb cget -_layout](#)

Exploring Ways to Obtain User-Supplied Data

In this procedure you use the Tcl `gets` command to capture the keyboard data, the `tk_getSaveFile` Tk command to create a file browser widget, and then configure the file browser widget.

Procedure

1. Create a file called *prompt.tcl* and add 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 explicitly specify 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 to display the “%” prompt.
 5. Type the following command and press Enter.

```
source prompt.tcl
```

When you source this script, it processes the first two lines, displaying the message and the prompt, then waits for your input.

6. Type:

```
sample_file.txt
```

Nothing occurs until you press Enter.

7. Press Enter.

The shell window prints the filename you entered. The full output in the shell window is as follows:

```
% 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.

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

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

9. Source the script in the terminal window to invoke a file browser dialog box.

```
source browser.tcl
```

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

10. Type in the filename “sample_file.txt” and click **Save**.
11. Check the terminal window. The script tells you what filename you entered:

```
% source browser.tcl
You entered /myDRV/examples/sample_file.txt
%
```

12. 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.

13. Save the file, then source it from the Tcl prompt.
14. Navigate to an actual file and select it, then click **Save**.

The tool 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 simplify writing code to manipulate files.

Writing a GUI Plug-in Procedure

In this procedure you learn about passing arguments to a GUI plug-in and then invoking the command procedure from the GUI plug-in.

Prerequisites

- Calibre DESIGNrev is running in interactive GUI mode and a layout is loaded

Procedure

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]
    puts "You entered $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 rarely use the second variable (window), but you must include it in the procedure definition; otherwise, the Tcl interpreter generates 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 is not 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.

Displaying a Message to the User

In this procedure you learn about the `tk_messageBox` command and using the backslash “\” to continue a line.

Prerequisites

- Calibre DESIGNrev is running in interactive GUI mode and a layout is loaded

Procedure

1. Experiment with the command that displays 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.

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 hierarchy was successfully \  
        written." -type ok  
}
```

Note the backslash “\” at the end of the line beginning with `tk_messageBox`. This indicates the command does 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 creates an error.

3. Save the file `write_hier.tcl`.
4. 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  
  
#####  
# COMMAND PROC: write_hier  
#####  
# Args:  
# wb = Calibre 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
}
```

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.

Related Topics

[\\$cwb cget -_layout](#)

[\\$L topcell](#)

[\\$L children](#)

Adding the Macro to the Menu

In this procedure you use the Macros create command to register the macro with a Calibre layout viewer and then use the Macros menu command to add the registered macro to the **Macros** menu.

Prerequisites

- Calibre DESIGNrev is running in interactive GUI mode and a layout is loaded

Procedure

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 appears 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, enter the following command:

```
source write_hier.tcl
```

4. In the shell window, enter the macros menu command:

```
Macros menu
```

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

Related Topics

[Macros create](#)

[Macros menu](#)

Building in Error Handling

The code you have written thus far displays a message box telling you the hierarchy was written successfully to the file. In this procedure you modify the Tcl script to include some error handling capability. You use the Tcl catch command to track whether a script returns an error, add conditional statements using if and if ... else, and then use the Tcl **return** command to exit the procedure.

Prerequisites

- Calibre DESIGNrev is running in interactive GUI mode and a layout is loaded

Procedure

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.

Because the Tcl *open* command used with the “w” argument 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 Tcl *catch* command tracks whether or not a script¹ 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 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* statement 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 *errMsg*. 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.

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.

- d. Select the file and click **Save**.
- e. When the macro displays a message box containing the error, click **OK**.

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 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*² for the procedure.

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

Replace the line:

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

With:

```
set return [write_hier $wbHandle $filename]  
if {$return eq "file_error"} {  
    return  
} else {  
    tk_messageBox -message "The layout hierarchy 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.

2. 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.

Setting Up Your Macro to Load Automatically

In this procedure you load macros in two different ways. Macros are loaded manually with the Macros menu command, or automatically by editing your configuration file.

Prerequisites

- Calibre DESIGNrev is running in interactive GUI mode and a layout is loaded

Procedure

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

```
calibredrv std_des.gds -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 *wbinit.tcl* 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 *wbinit.tcl* file before creating the menus.

Related Topics

[Macros menu](#)

Sample Macro Code

The complete code example of the *write_hier.tcl* file used in procedures is provided.

Figure 4-1. Completed Code (write_hier.tcl)

```
#####
# COMMAND PROC: write_hier
#####
# Args:
# wb = Calibre 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
    }
}

if {[info exists tk_version]} {
    Macros create "my macro" write_hier write_hier_plug_in
}
```

Related Topics

[\\$cwb cget - _layout](#)

[\\$L children](#)

[\\$L topcell](#)

[Macros create](#)

Macro Examples to Study and Learn

You can learn more about creating macros by studying some existing macros.

List All Layers	92
Draw a Circle	92
DRC External Spacing Check	94

List All Layers

In this example, a simple macro command proc and plug-in demonstrate the syntax. The following Tcl procedure defines a macro that lists all layers present in the current layout viewer window. The layout peek command can also list layers.

Example 4-1. List All Layers

```
# INPUT: wbHandle - Handle to a Calibre WORKbench object
# window - Window name for Calibre WORKbench object
# OUTPUT: Prints 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
#
if {[info exists tk_version]} {
    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.

Draw a Circle

The macro in this example is used to draw a circle. It creates a GUI interface that prompts for the information (radius, number of sides, and layer) that is used to draw the circle.

Example 4-2. Draw a Circle

```

#
# Macro to draw a circle
#

proc ui_circle {wbHandle window} {
    set L0 [ $wbHandle cget -_layout ]
    if { $L0 == "" } {
        error "Cannot continue without an open layout"
    }

    #
    #   creates a gui to enter the parameters
    #

    global cancel
    global runme
    set cancel "0"
    set runme "0"
    global expand
    global f
    set f .expand
    eval {toplevel $f} -borderwidth 10
    wm title $f "Draw Circle"
    message $f.msg -text "Enter Circle Information" -aspect 1000
    label $f.lradius -text "Radius:" -anchor w
    entry $f.eradius -textvariable expand(radius) -relief sunken
    label $f.lsides -text "Sides:" -anchor w
    entry $f.esides -textvariable expand(sides) -relief sunken
    label $f.llayer -text "Layer:" -anchor w
    entry $f.elayer -textvariable expand(layer) -relief sunken
    set b [frame $f.buttons]
    pack $f.msg $f.lradius $f.eradius $f.lsides $f.esides $f.llayer \
        $f.elayer $f.buttons -side top -fill x
    button $b.ok -text Run -command {cancel_and_run}
    bind $f.eradius <Return> {cancel_and_run}
    bind $f.esides <Return> {cancel_and_run}
    bind $f.elayer <Return> {cancel_and_run}
    button $b.cancel -text Cancel -command {cancel_and_return}
    pack $b.ok -side left
    pack $b.cancel -side right
    tkwait window $f

    if { $cancel == 1 || $runme == 0 } {
        return
    }

    circle $L0 $expand(radius) $expand(sides) $expand(layer)
    $L0 update
    $wbHandle updateDisplay
    puts "Draw Circle Completed \n"

    tk_messageBox -message "Draw Circle Completed" -type ok
}

proc cancel_and_return { } {
    global f
    global cancel

```

```
        set cancel "1"
        destroy $f
    }

    proc cancel_and_run { } {
        global f
        global runme
        set runme "1"
        destroy $f
    }

    proc circle {L radius sides layer} {
        set precision [ $L units microns ]
        for {set i 1} {$i <= $sides} {incr i} {
            set x [expr { [format %2.2f $radius] * \
                cos(($i/[format %2.2f $sides] * 2 * 3.14159265 ))}]
            set y [expr { [format %2.2f $radius] * \
                sin(($i/[format %2.2f $sides] * 2 * 3.14159265 ))}]
            append coords "[expr int($precision * [format %2.3f $x])] \
                [expr int($precision * [format %2.3f $y])] "
        }
        set cmd "$L create polygon TOPCELL $layer $coords"
        eval $cmd
        return
    }

    if { [ isTkLoaded ] } {
        Macros create "DRAW CIRCLE" circle ui_circle
    }
}
```

DRC External Spacing Check

This example is more elaborate, and therefore better represents the usefulness of macro extensions. 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 *wbinit.tcl* file.

Example 4-3. DRC External Spacing Check

```
#
# INPUT: wbHandle - Handle to a Calibre WORKbench object
# window - Window name for Calibre WORKbench object
# OUTPUT: Prints 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
if {[info exists tk_version]} {
    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. However, complex macros are easier to build and debug when created as two separate procedures.

A 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:

```
if {[info exists tk_version]} {
    macros create {macro name} my_macro my_macro
}
```

The **macros create** Tcl command is only available when running the layout viewer's GUI, and not when running in shell mode. Therefore, the previous call is wrapped in an if block to avoid errors when the macro is sourced.

Chapter 5

Layout Viewer Batch Commands and Object Methods

This chapter contains reference pages for the layout viewer batch commands and object methods. The reference pages are organized into sections based on function (command or object method), and the object type they are designed to operate on (cwbWorkBench, layout, overlay, peek, cwbLayoutView, and StringFeature). This chapter includes reference pages for Macro commands that you use to create, delete, and manipulate macros in the layout viewers.

The commands and object methods documented in this chapter are supported in Calibre DESIGNrev, Calibre WORKbench, Calibre LITHOview, and Calibre MDPview unless otherwise noted. The commands and object methods are supported in High Capacity (HC) mode unless otherwise noted in the reference pages. Refer to “[High Capacity Mode](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for more information on HC mode.

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.

Unsupported Commands and Arguments in High Capacity (HC) Mode	100
cwbWorkBench Commands	101
cwbWorkBench Object Methods	104
Layout Commands	152
Layout Object Methods	235
Overlay Commands	437
Overlay Objects Methods	446
Peek Object Methods	453
StringFeature Commands	464
StringFeature Object Methods	467
cwbLayoutView Object Methods	469
Macros Commands	476
Miscellaneous Batch Commands	483

Unsupported Commands and Arguments in High Capacity (HC) Mode

Some batch commands and arguments are not supported in HC mode.

Table 5-1. Unsupported Commands and Arguments in HC Mode

Command	Status
layout copy	Not supported.
layout copy2	Not supported.
layout create (GDS or OASIS file)	The -pcr_file and -validate_pcr arguments are not supported.
layout create (mapped layout)	The -pcr_file, -validate_pcr, and -log arguments are not supported.
layout create (multiple layouts)	The -log argument is not supported.
layout merge	Not supported.
layout tbmode	The extent argument is not supported.
\$L COPYCELL GEOM	Not supported.
\$L import layout	Not supported.
\$L gdsout	The -place and -texttype arguments are not supported.
\$L oasisout	The -place argument is not supported.
\$L property	Not supported.

cwbWorkBench Commands

Use the cwbWorkBench commands to manipulate database files.

Table 5-2. cwbWorkBench Command Summary

cwbWorkBench Commands	Description
cwbLoadRdbFileInRve	Opens the specified ASCII format DRC results database (RDB) in Calibre RVE.
cwbOpenDBInRve	Opens the specified results database in Calibre RVE.

cwbLoadRdbFileInRve

Opens the specified ASCII format DRC results database (RDB) in Calibre RVE.

Usage

cwbLoadRdbFileInRve *rdbFileName*

Arguments

- *rdbFileName*

A required argument that specifies the path to an ASCII format results database.

Description

This command opens an ASCII format results database (RDB) in Calibre RVE. If a Calibre RVE session is already running, the RDB file is opened in the existing session. Otherwise, this command starts a Calibre RVE session and opens the RDB file.

Examples

```
cwbLoadRdbFileInRve net.db
```

cwbOpenDBInRve

Opens the specified results database in Calibre RVE.

Usage

cwbOpenDBInRve *dbname* [*topcell*]

Arguments

- *dbname*
A required argument specifying the name of the results database to open in Calibre RVE. The database name can be a file or a directory. The supported database types include DRC (ASCII format), PERC, PEX, LVS, DFM, and SPICE.
- *topcell*
An optional argument specifying the name of the top cell. This argument is only valid for LVS, PEX, and PERC databases.

Description

This command opens the specified database in an existing Calibre RVE session or starts a Calibre RVE session if one is not already running. Any previously opened LVS, PERC, PEX, and DFM databases are closed before opening new databases. A DRC database is opened in a new tab when DRC databases are already opened in Calibre RVE. SPICE files are opened in the SPICE browser.

Examples

```
cwbOpenDBInRve layout.drc.results
```

cwbWorkBench Object Methods

Use the cwbWorkBench object methods to configure or access information about the current layout viewer session object and other state-dependent data.

A cwbWorkBench object is referenced by its handle, which also acts as a command for operations performed on the current layout in the current layout viewer session. The command reference pages in this section use the \$cwb variable to refer to the cwbWorkBench handle. You must assign a value to this variable name when executing the command. The cwbWorkBench object methods described in this section can be used for all layout viewers.

Table 5-3. cwbWorkBench Object Methods Summary

cwbWorkBench Object Methods	Description
\$cwb bindKey	Sets a key or mouse wheel binding.
\$cwb cget - _layout	Returns the handle of the database object visible in the layout viewer main window.
\$cwb cget - _layoutView	Gets the handle of the layoutView object for the specified cwbWorkBench object.
\$cwb deleteLayoutClbk	Closes the currently displayed layout and deletes the layout handle.
\$cwb exportLayerPalette	Exports the layer palette as an image file.
\$cwb getMenuPath	Gets the handle of the menu for the specified cwbWorkBench object.
\$cwb getRulerUnits	Gets the ruler units.
\$cwb getSelectionIterator	Gets the iterator for the current selection.
\$cwb getSelectionPoint	Gets the coordinates of the selection point.
\$cwb getSelectionRect	Gets the lower-left and upper-right coordinates of the selection rectangle.
\$cwb getVisibleLayers	Gets the numbers of the visible layers.
\$cwb hasSelectionPoint	Determines if a selection point is set.
\$cwb loadDefaultLayerProperties	Loads or reloads the default layer properties file.
\$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.

Table 5-3. cwbWorkBench Object Methods Summary (cont.)

cwbWorkBench Object Methods	Description
\$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.
\$cwb setLayerVisibility	Sets the visibility for one or more layers.
\$cwb setReferenceVisibility	Sets the visibility to show or hide one or more cell references in a layout.
\$cwb show	Redisplays the layout.
\$cwb showWithLayerFilterClbk	Redraws the display with a filtered set of layers.
\$cwb synchronizeWindowView	Preserves the current viewed coordinates when switching the cell or layout that is being viewed.
\$cwb unsynchronizeWindowView	Disables the preserving of viewing coordinates when switching cells or layouts.
\$cwb updateDisplay	Redraws the display.
\$cwb viewedLayoutClbk	Changes the currently-displayed layout to a specified layout object.
\$cwb waitForDrawing	Measures the time it takes for the last-issued drawing command to complete.
\$cwb zoomAllClbk	Fills the display with the entire layout design.
\$cwb zoomToRectClbk	Zooms the window to the specified coordinates.

\$cwb bindKey

Sets a key or mouse wheel binding.

Usage

\$cwb bindKey <keyname> {*cwbAction* | *user_proc* | **null**} [*menu_name* "menu_pick"]

Arguments

- <keyname>

A required argument specifying the name of a key or mouse wheel combination. Refer to [Table 5-4](#) for a list of the supported key and mouse wheel bindings. The angle brackets (< >) around the keyname are required. Any key combinations not customized remain as their default values.

- *cwbAction* | *user_proc* | **null**

A required argument specifying the action to associate with the <keyname>. Valid values include the following:

cwbAction — Specifies the name of an action to execute. Refer to [Table 5-5](#) for a list of actions that you can bind to a key or the mouse wheel. If you want to associate the action with a menu, you must specify values for the *menu_name* and "menu_pick".

user_proc — Specifies the name of a user process to execute. This can be a Tcl script you have defined and loaded.

null — Removes the key or mouse wheel binding. If the binding appears in a menu, you must specify values for the *menu_name* and "menu_pick" arguments to remove the binding from the menu.

- *menu_name*

An optional argument specifying the name of the menu (for example, **File** or **Edit**) in which you want to add or remove the binding. When you create a new key or mouse wheel binding, the application also automatically removes the label for the new binding from any menu item that previously used it.

Note



This argument only adds or removes custom key or mouse wheel bindings and cannot be used to change the core set of menu items available in the GUI.

- "menu_pick"

An optional argument specifying the text string that appears in the menu specified by *menu_name*. The quotes (" ") are required. When removing an item from a menu, this value must exactly match the item that appears in the menu.

Return Values

A summary of your new key binding.

Description


You can use the \$cwb bindKey command in a Tcl script or procedure to create a custom key or mouse wheel binding. The predefined set of supported Tcl key and mouse wheel combinations is listed in [Table 5-4](#). Other Tcl variants for the same key or mouse wheel combinations are not recognized by the layout viewers.

You can also define or modify key and mouse wheel bindings in a *keyprefs* file. Refer to [keyprefs File Format](#) in the *Calibre DESIGNrev Layout Viewer User's Manual* for information on creating a *keyprefs* file.

Table 5-4. Supported Key and Mouse Wheel Combinations for Custom Key Bindings

Key Type	Key Range
Alt keys	<Alt-A> through <Alt-Z> <Alt-a> through <Alt-z> <Alt-Key-0> through <Alt-Key-9>
Control keys	<Control-A> through <Control-Z> <Control-a> through <Control-z> <Control-Key-0> through <Control-Key-9> <Control-comma> <Control-period> <Control-Down> <Control-Up> <Control-Left> <Control-Right>
Function keys	<Key-F1> through <Key-F12> <Key-F24> <Control-F1> through <Control-F12> <Shift-F1> through <Shift-F12>
Regular keys	<Key-A> through <Key-Z> <Key-a> through <Key-z> <Key-0> through <Key-9>

Table 5-4. Supported Key and Mouse Wheel Combinations for Custom Key Bindings (cont.)

Key Type	Key Range
Special keys	<Key-Backspace> <Key-Delete> <Key-Escape> <Key-KP_Subtract> <Key-KP_Add> <Key-Down> <Key-Left> <Key-Right> <Key-Up>
Mouse wheel	<Button-4> through <Button-5>  Note: Button-4 and Button-5 scroll up and down, respectively. <Shift-Button-4> through <Shift-Button-5> <Control-Button-4> through <Control-Button-4> <Alt-Button-4> through <Alt-Button-5> <Shift-Control-Button-4> through <Shift-Control-Button-5>

Note


 Keys <Control-F1> through <Control-F12> as well as <Shift-F1> through <Shift-F12> are reserved, and therefore are not set by the layout viewers.

Table 5-5 lists the actions you can bind to a key.

Table 5-5. Actions You Can Bind To Keys

Action	Action Performed
cwbCancel	Cancels the current dialog box.
cwbClearContext	Unsets the selected cell reference as the editing context and returns the editing context to the topcell of the layout.
cwbClearRulerAll	Clears all rulers.
cwbClearRulerLast	Clears the last ruler drawn.
cwbCloseLayout	Closes the currently active layout. If this was the last layout in the viewer, the window exits.

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

Action	Action Performed
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.
cwbLoadDefaultLayerProperties	Loads the default layer properties for the layout viewer.
cwbMacrosCutRegion	Runs the CUT REGION macro.

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

Action	Action Performed
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.
cwbPreferences	Opens the Preferences dialog box.

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

Action	Action Performed
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.

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

Action	Action Performed
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.
cwbUnselectAll	Unselects all selections.
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.

Examples

Example 1

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"
```

Example 2

This next example erases the F1 key binding, 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 Topics

[keyprefs File Format \[Calibre DESIGNrev Layout Viewer User's Manual\]](#)

\$cwb cget -_layout

Returns the handle of the database object visible in the layout viewer main window.

Usage

\$cwb cget -_layout

Arguments

- **-_layout**

A required keyword specifying to return the handle of the current database object.

Return Values

The handle of the current layout or overlay, such as “layout0” or “overlay0”.

Description

Gets the handle of the database object currently visible in the Calibre layout viewer main window. This command returns only one handle, even if you have loaded multiple layouts or overlays into the layout viewer.

When writing scripts or issuing commands at the Tcl prompt, you can get the cwbWorkBench handle using the **find** command:

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

Examples

This example sets the cwb variable to the first cwbWorkBench handle returned by the getCWBOObjects command. It then sets the my_layout variable to the handle of the current layout and returns a list of all cells in the layout.

```
% set cwb [lindex [getCWBOObjects] 0]
::cwbWorkBench::cwbWorkBench0
% set my_layout [$cwb cget -_layout]
layout0
% $my_layout cells
TOPCELL a463 a310 b561 c980 981
```

\$cwb cget -_layoutView

Gets the handle of the layoutView object for the specified cwbWorkBench object.

Usage

\$cwb cget -_layoutView

Arguments

- **-_layoutView**

A required keyword specifying to return the handle of the layoutView object, which is a Tk object that controls the display of layout data.

Return Values

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.

Description

Gets the handle of the layoutView object for the specified cwbWorkBench object.

Examples

This example sets the wb variable to the cwbWorkBench handle returned by the find command. It then uses the wb variable to return the handle of the layoutView.

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

\$cwb deleteLayoutClbk

Closes the currently displayed layout and deletes the layout handle.

Usage

\$cwb deleteLayoutClbk

Arguments

None.

Return Values

None.

Description

Closes the currently displayed layout and deletes the layout handle. If there have been edits, the tool asks if you want to save your layout changes. Use this command in scripts working with a displayed layout in the GUI.

Examples

This example sets the wb variable to the cwbWorkBench handle returned by the find command and then closes the layout and deletes the layout handle.

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

Related Topics

[\\$cwb reloadLayoutClbk](#)

[\\$cwb viewedLayoutClbk](#)

[layout delete](#)

\$cwb exportLayerPalette

Exports the layer palette as an image file.

Usage

\$cwb exportLayerPalette *imagetype filename*

Arguments

- *imagetype*

A required argument specifying the image type to output. Valid values include:

- *jpg* — Joint Photographic Experts Group (JPG) is a lossy compression format which supports 8-bit grayscale images and 24-bit color images.
- *png* — Portable Network Graphics (PNG) is both a lossy and lossless compression format which supports 8-bit paletted images and 24-bit color images.
- *gif* — Graphics Interchange Format (GIF) is a lossless compression format limited to an 8-bit palette, or 256 colors.
- *bmp* — Windows bitmap file format (BMP) is an uncompressed format.
- *xpm* — X Pixmap (XPM) is a bitmap format.
- *ppm* — Portable Pixmap file format (PPM) is a bitmap format.
- *tiff* — Tagged Image File Format (TIFF) is both a lossy and lossless compression format which supports 8-bit or 16-bit per color (red, green, blue) images.

- *filename*

A required argument specifying the filename to output the image to. The specified filename can include a directory path. If a file already exists by that name in the destination directory, it is overwritten.

Return Values

None.

Description

Exports the layout viewer's layer palette as an image file. This snapshot is only an image of the list of layers displayed in the layer palette. If the list of layers is longer than the vertical length of the layer palette, the image is truncated and only contains displayed layers.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% $wb exportLayerPalette jpg mylayers.jpg
% ls
mylayers.jpg
```

\$cwb getMenuPath

Gets the handle of the menu for the specified cwbWorkBench object.

Usage

\$cwb getMenuPath

Arguments

None.

Return Values

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

Description

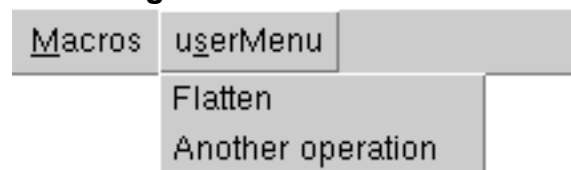
Gets the handle of the menu path for the specified cwbWorkBench object. Use this function to assist in adding menus to your layout viewer menu bar.

Examples

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 5-1. Added Menu



\$cwb getRulerUnits

Gets the ruler units.

Usage

\$cwb getRulerUnits

Arguments

None.

Return Values

Returns the ruler units. Possible return values are:

- um — microns
- nm — nanometers
- dbu — database units

Description

Gets the ruler units. You can set the ruler units using the Units preference on the **Ruler** tab of the Preferences dialog box.

Examples

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

Related Topics

[\\$cwb rulerMeasurements](#)

\$cwb getSelectionIterator

Gets the iterator for the current selection.

Usage

\$cwb getSelectionIterator

Arguments

None.

Return Values

Returns an iterator for the current selection.

Description

Retrieves the iterator for the current selection. The layout viewer selected objects can be looped through using the selection iterator.

Each selection result is returned as a Tcl associative array. The returned Tcl array contains relevant key names for each object type:

- Path (wire)
 - type = “wire”
 - beginExtension
 - endExtension
 - layer
 - outlinePoints
 - path
 - pathType
 - points
 - properties
 - width
- Polygon
 - type = “polygon”
 - layer
 - path
 - points

- properties
- Reference
 - type = “ref”
 - angle
 - cell
 - columns
 - height
 - llx
 - lly
 - magnification
 - mirrorX
 - path
 - properties
 - rows
 - width
 - x
 - xSpacing
 - y
 - ySpacing
- Text
 - type = “text”
 - angle
 - layer
 - magnification
 - path
 - presentation
 - properties
 - string
 - transformation

- x
- y
- Edge
 - type = “edge”
 - edgeIndex
 - path
 - In addition, all keys from the parent path or polygon.
- Vertex
 - type = “vertex”
 - path
 - vertexIndex
 - In addition, all keys from the parent path or polygon.

Note



Edge and vertex types are selections of a parent path or polygon, therefore their results include the parent's key names and values except for the parent type.

Examples

Example 1

This example assumes two polygons, a path, and a reference are selected. The information for the selected objects is output to the transcript:

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set iter [$cwb getSelectionIterator]
egds_Selection_0_iterator

% # Print out first object, a path
% $iter next
type wire layer 4 width 2000 pathType 0 beginExtension 0 endExtension 0
points {{x -86000 y -58000} {x -59000 y -58000} {x -59000 y -24000}
{x -35000 y -24000}} outlinePoints {{x -86000 y -59000}
{x -58000 y -59000} {x -58000 y -25000} {x -35000 y -25000}
{x -35000 y -23000} {x -60000 y -23000} {x -60000 y -57000}
{x -86000 y -57000}} path /mix}

% # Print out second object, a polygon
% $iter next
type polygon layer 3 points {{x -108000 y 6000} {x -81000 y 6000}
{x -81000 y 20000} {x -108000 y 20000}} path /mix/inv}

% # Print out third object, another polygon
% $iter next
type polygon layer 5 points {{x -108000 y -16000} {x -81000 y -16000}
{x -81000 y -1000} {x -108000 y -1000}} path /mix/inv}

% # Print out fourth and final object, a reference
% $iter next
type ref llx -40000 lly -102000 width 70000 height 122000 cell nand
x 0 y -22000 mirrorX 0 angle 0.0 magnification 1.0 path /mix}
```

Example 2

In this example, the selection iterator outputs information for the selected objects from the Tcl associative array:

```
% set cwb [find objects -class cwbWorkBench]
% set iter [$cwb getSelectionIterator]
% for {set result [$iter next]} {0 < [llength $result]} \
    {set result [$iter next]} {
    array unset results
    array set results $result
    foreach name [array names results] {
        puts "iterator\[ $results(type) \] \[ $name \] = $results($name) "
    }
    puts ""
}

iterator[wire][endExtension] = 0
iterator[wire][width] = 2000
iterator[wire][path] = /mix
iterator[wire][outlinePoints] = {x -86000 y -59000} {x -58000 y -59000}
    {x -58000 y -25000} {x -35000 y -25000} {x -35000 y -23000}
    {x -60000 y -23000} {x -60000 y -57000} {x -86000 y -57000}
iterator[wire][points] = {x -86000 y -58000} {x -59000 y -58000}
    {x -59000 y -24000} {x -35000 y -24000}
iterator[wire][layer] = 4
iterator[wire][beginExtension] = 0
iterator[wire][type] = wire
iterator[wire][pathType] = 0

iterator[polygon][path] = /mix/inv
iterator[polygon][points] = {x -108000 y 6000} {x -81000 y 6000}
    {x -81000 y 20000} {x -108000 y 20000}
iterator[polygon][layer] = 3
iterator[polygon][type] = polygon

iterator[polygon][path] = /mix/inv
iterator[polygon][points] = {x -108000 y -16000} {x -81000 y -16000}
    {x -81000 y -1000} {x -108000 y -1000}
iterator[polygon][layer] = 5
iterator[polygon][type] = polygon

iterator[ref][llx] = -40000
iterator[ref][lly] = -102000
iterator[ref][height] = 122000
iterator[ref][magnification] = 1.0
iterator[ref][type] = ref
iterator[ref][angle] = 0.0
iterator[ref][width] = 70000
iterator[ref][cell] = nand
iterator[ref][x] = 0
iterator[ref][y] = -22000
iterator[ref][mirrorX] = 0
iterator[ref][path] = /mix
```

Related Topics

[\\$cwb getSelectionPoint](#)

[\\$cwb getSelectionRect](#)

\$cwb getSelectionPoint

Gets the coordinates of the selection point.

Usage

\$cwb getSelectionPoint [microns | dbu]

Arguments

- microns | dbu

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

Return Values

Returns the coordinates of the selection point. If no selection point exists, returns a list containing the coordinates -0.001 -0.001, or -1 -1 if dbu is specified.

Description

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

Examples

The following example checks whether the cwbWorkBench object has a selection point, then gets the coordinates of the selection point (in database units):

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

Related Topics

[\\$cwb hasSelectionPoint](#)

\$cwb getSelectionRect

Gets the lower-left and upper-right coordinates of the selection rectangle.

Usage

\$cwb getSelectionRect [microns | dbu]

Arguments

- microns | dbu

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

Return Values

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

Description

Returns a string containing the lower left and upper right coordinates of the analysis rectangle drawn in the View Center window (the selection rectangle). If there is no selection rectangle, the command returns the coordinates of the viewed window.

Examples

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

\$cwb getVisibleLayers

Gets the numbers of the visible layers.

Usage

\$cwb getVisibleLayers

Arguments

None.

Return Values

Returns the visible layer numbers in the layer palette.

Description

Returns all layers set to visible in the layer palette for the current layout. The command operates on layouts, overlays, and MDP jobdecks. Operates with either filtered mode (layer filter active) or net extraction filtering.

Examples

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

Related Topics

[\\$L layers](#)

[\\$cwb setLayerVisibility](#)

\$cwb hasSelectionPoint

Determines if a selection point is set.

Usage

\$cwb hasSelectionPoint

Arguments

None.

Return Values

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

Examples

This example demonstrates the return values of calling \$cwb hasSelectionPoint with no selection point, and then calling it again after a selection point has been made.

```
% set cwb [find objects -class cwbWorkBench]
% $cwb hasSelectionPoint
0
% $cwb hasSelectionPoint
1
% $cwb getSelectionPoint dbu
32000 45000
```

Related Topics

[\\$cwb getSelectionPoint](#)

\$cwb loadDefaultLayerProperties

Loads or reloads the default layer properties file.

Usage

\$cwb loadDefaultLayerProperties

Arguments

None.

Return Values

None.

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.

Examples

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

Related Topics

[\\$cwb loadLayerProperties](#)

\$cwb loadLayerProperties

Loads layer properties from a specified file.

Usage

\$cwb loadLayerProperties *filename*

Arguments

- *filename*

A required argument specifying the filename of a layer properties file to load.

Return Values

None.

Description

Loads or reloads the specified layer properties file. The command generates an error if the layer properties file does not exist or is unreadable.

Examples

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

Related Topics

[\\$cwb loadDefaultLayerProperties](#)

[Layerprops File Format \[Calibre DESIGNrev Layout Viewer User's Manual\]](#)

\$cwb reloadLayoutClbk

Reloads the currently displayed layout.

Usage

\$cwb reloadLayoutClbk

Arguments

None.

Return Values

None.

Description

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

Examples

This example reloads the displayed layout.

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

Related Topics

[\\$cwb deleteLayoutClbk](#)

\$cwb rulerMeasurements

Shows information on rulers in the layout.

Usage

\$cwb rulerMeasurements

Arguments

None.

Return Values

Returns information for rulers in the following format:

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

Description

Displays ruler information for rulers in the specified layout. If specified on the command line, it displays information for all active rulers.

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
Dy -3755 Distance 3755.0
```

A more advanced version of the example:

```
foreach ruler [$cwb 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 Topics

[\\$cwb getRulerUnits](#)

\$cwb runMultiRveOnRveOutput

Starts Calibre RVE, without closing existing sessions.

Usage

\$cwb runMultiRveOnRveOutput [*databasefile*]

Arguments

- *databasefile*

An optional argument specifying a database file to load. If this argument is not specified, Calibre RVE prompts for a filename when the tool loads.

Return Values

None.

Description

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

Examples

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

Related Topics

[\\$cwb runRveOnRveOutput](#)

\$cwb runRveOnRveOutput

Starts Calibre RVE, a graphical debug program that interfaces with most IC layout tools.

Usage

\$cwb runRveOnRveOutput [*databasefile*]

Arguments

- *databasefile*

An optional argument specifying a database file to load. If this argument is not specified, Calibre RVE displays a dialog box prompting for the name of the database file to load.

Return Values

None.

Description

Starts a Calibre RVE session, and outputs the fact that the layout viewer is loading Calibre RVE. Calibre layout viewers are designed to be used in conjunction with the Calibre RVE graphical debug program when running checks and reviewing check results from most of the Calibre batch tools.

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

Note



You can only open one Calibre RVE window with this command.

Examples

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

Related Topics

[\\$cwb runMultiRveOnRveOutput](#)

[\\$cwb show](#)

\$cwb selectionCloneToNewLayout

Copies all selected geometry to a new layout object.

Usage

\$cwb selectionCloneToNewLayout

Arguments

None.

Return Values

The new layout handle, such as “layout3”.

Description

Copies all selected polygons, paths, and text to a new layout object, and displays the handle to the new layout object. The topcell name stays the same, and it contains all cloned geometries, including path types. References are not copied.

Examples

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


\$cwb setDepth

Sets the depth for the Calibre layout viewer.

Usage

\$cwb setDepth *startDepth endDepth*

Arguments

- ***startDepth***
A required argument that specifies the starting depth to use for the Calibre layout viewer.
- ***endDepth***
A required argument that specifies the ending depth to use for the Calibre layout viewer.

Return Values

None.

Description

Sets the starting and ending depth for the Calibre layout viewer. You need to follow this command with a \$cwb updateDisplay command to see the effects of the command in GUI mode.

Examples

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

Related Topics

[\\$cwb updateDisplay](#)

\$cwb setLayerVisibility

Sets the visibility for one or more layers.

Usage

\$cwb setLayerVisibility *layers operation*

Arguments

- **layers**
A required argument specifying a layer number, a list of layer numbers, or the name of a layer filter.
- **operation**
A required argument specifying the operation to be performed by the command. Valid values include:
 - show — Shows the specified layers.
 - hide — Hides the specified layers.
 - toggle — Toggles the visibility of the specified layers. Layers that are currently hidden are shown, and layers that are currently shown are hidden.

Return Values

None.

Description

Sets the visibility for one or more layers in a layout, overlay, or MDP jobdeck. Layers are specified as a layer number, a list of layer numbers, or as the name of a layer filter. This command generates a warning when a specified layer does not exist. For example:

```
Warning: Can not change the visibility of layer "10", it does not exist in
the layout.
```

Examples

Example 1

In this example, the script toggles the visibility of layers 1, 4, and 5 in the layout *lab1a.gds*.

```
set cwb [find objects -class cwbWorkBench]
set L [layout create ./lab1a.gds]
$cwb viewedLayoutClbk $L
$cwb setLayerVisibility [list 1 4 5] toggle
```

Example 2

This example does the following:

- Defines a layers filter in a *layerprops* file for the layout named *my_layout.gds*. In this example, the *layerprops* file is located in the same directory as the layout.

- Creates Tcl procs in the *wbinit.tcl* file that use \$cwb setLayerVisibility to toggle the visibility of layers 1, 3, and the “baselayers” layers filter.
- Adds the bindKey command to the *keyprefs* file to bind the Tcl procs to keystrokes.

1. Create the following layer property file.

```
0 green speckle 0 1 1
1 blue speckle 1 1 1
1.1 blue speckle 1.1 1 1
1.2 blue speckle 1.2 1 1
2 yellow speckle 2 1 1
2.1 yellow speckle 2.1 1 1
3 red speckle 3 1 1
3.1 red speckle 3.1 1 1
3.2 red speckle 3.2 1 1
layerFilters -add baselayers 1.1 2.1 3.1
```

2. Save the layer property file as *<layout_path>/my_layout.gds.layerprops*.

3. Edit the *\$HOME/.calibrewb_workspace/wbinit.tcl* file and add the following:

```
proc toggle_metal_1 {} {
    set CWB [lindex [getCWBObjects] 0]
    $CWB setLayerVisibility 1 "toggle"
}
proc toggle_metal_2 {} {
    set CWB [lindex [getCWBObjects] 0]
    $CWB setLayerVisibility 3 "toggle"
}
proc toggle_all_base_layers {} {
    set CWB [lindex [getCWBObjects] 0]
    $CWB setLayerVisibility "baselayers" "toggle"
}
```

Save and close the file.

4. Edit the *\$HOME/.calibrewb_workspace/keyprefs* file and add the following:

```
bindKey <Key-1>      toggle_metal_1
bindKey <Key-9>      toggle_metal_2
bindKey <Key-0>      toggle_all_base_layers
```

Save and close the file.

5. Open the layout *my_layout.gds* in Calibre DESIGNrev.

This automatically loads the layer properties file (*my_layout.gds.layerprops*) and the *\$HOME/.calibrewb_workspace/wbinit.tcl* file. You can then use keys 1, 9, and 0 to toggle the visibility of layers in the layout.

Related Topics

[\\$cwb getVisibleLayers](#)

[\\$L layerFilters](#)

[Layerprops File Format \[Calibre DESIGNrev Layout Viewer User's Manual\]](#)

\$cwb setReferenceVisibility

Sets the visibility to show or hide one or more cell references in a layout.

Usage

\$cwb setReferenceVisibility *cells operation*

Arguments

- **cells**
A required argument specifying the name of one or more cells for setting the visibility of the references. Cells are specified as a cell name or a list of cell names. Cell names can be an explicit name or a wildcard. Supported wildcards include the asterisk (*) that matches 0 or more characters, or a question mark (?) that matches exactly one character.
- **operation**
A required argument specifying the operation to perform on the references. Valid values include:
 - show — Shows the references for the specified cell(s).
 - hide — Hides the references for the specified cell(s).
 - toggle — Toggles the visibility of the references for the specified cell(s). References that are currently hidden are shown, and references that are currently shown are hidden.

Return Values

None.

Description

Sets the visibility to show or hide one or more cell references in a layout. This command outputs a warning if you specify a top cell, either explicitly or with a wildcard match, and proceeds with other cells. For example:

```
Warning: ignoring top cell "top".
```

This command generates a warning when a specified cell does not exist. For example:

```
Warning: cell "abcd" not found in layout "layout0" : "fullchip.oas".
```

Examples

Example 1

This Tcl script loads the layout named *lab1a.oas* and hides references to cells named “a9500” and “a2311”.

```
set cwb [find objects -class cwbWorkBench]
set L [layout create ./lab1a.oas]
$cwb viewedLayoutClbk $L
$cwb setReferenceVisibility [list a9500 a2311] hide
```

Example 2

This Tcl script loads the layout named *fullchip.oas* and toggles the visibility of cell references whose name begins with “INV” or “BUFX”. The asterisk (*) matches 0 or more characters for “INV” cells, while the question mark (?) matches exactly one character for “BUFX” cells.

```
set cwb [find objects -class cwbWorkBench]
set L [layout create ./fullchip.oas]
$cwb viewedLayoutClbk $L
# Toggle the visibility of references for INV* and BUFX? cells
$cwb setReferenceVisibility [list INV* BUFX?] toggle
```

Example 3

This Tcl script loads the layout named *fullchip.oas* and hides all cell references in the layout. The script then shows only the references for the cells whose name begins with “PAD*”. The asterisk (*) matches 0 or more characters for cell names that begin with “PAD” (for example, PADCORNER, PADGND, and PADINC).

```
set cwb [find objects -class cwbWorkBench]
set L [layout create ./fullchip.oas]
$cwb viewedLayoutClbk $L

# Hide all references
$cwb setReferenceVisibility [list *] hide

# Show only references for cells named PAD*
$cwb setReferenceVisibility [list PAD*] show
```

\$cwb show

Redisplays the layout.

Usage

\$cwb show

Arguments

None.

Return Values

Redisplays the layout.

Description

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

Examples

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

Related Topics

[\\$cwb updateDisplay](#)

\$cwb showWithLayerFilterClbk

Redraws the display with a filtered set of layers.

Usage

\$cwb showWithLayerFilterClbk [*layer_filter*]

Arguments

- *layer_filter*

An optional argument specifying the name of the layer filter designator.

Return Values

None.

Description

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

Examples

Example 1

This example creates a layer filter named “my_layers” for *lab1a.gds* and adds layers 2, 4, and 6 to the filter. The script then uses the \$cwb showWithLayerFilterClbk command to redraw the display and show only layers 2, 4, and 6. The visible layers are then output to *output.txt*.

```
set cwb [find objects -class cwbWorkBench]
set L [layout create lab1a.gds]
$L layerFilters -add my_layers 2 4 6
$cwb viewedLayoutClbk $L
set results [open ./output/output.txt w]
$cwb showWithLayerFilterClbk my_layers
puts $results [$cwb getVisibleLayers]
close $results
```

Related Topics

[\\$L layerFilters](#)

\$cwb synchronizeWindowView

Preserves the current viewed coordinates when switching the cell or layout that is being viewed.

Usage

\$cwb synchronizeWindowView

Arguments

None.

Description

This command causes the currently open window views to be synchronized with each other. It is functionally equivalent to enabling the “Preserve view coordinates” pushpin button located in the lower-left corner of the GUI. Opening a new window after using this method requires the command to be executed again in order to synchronize the new window.

Examples

The following example synchronizes all existing window views.

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% foreach cwb [getCWBObjects] {
    $cwb synchronizeWindowView
}
```

\$cwb unsynchronizeWindowView

Disables the preserving of viewing coordinates when switching cells or layouts.

Usage

\$cwb unsynchronizeWindowView

Arguments

None.

Description

This command causes the currently open window views to not be synchronized with each other. It is functionally equivalent to disabling the “Preserve view coordinates” pushpin button located in the lower-left corner of the GUI. Opening a new window after using this method requires the command to be executed again in order to unsynchronize the new window.

Examples

The following example unsynchronizes all existing window views.

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% foreach cwb [getCWBObjects] {
    $cwb unsynchronizeWindowView
}
```

\$cwb updateDisplay

Redraws the display.

Usage

\$cwb updateDisplay

Arguments

None.

Return Values

None.

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.

Examples

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

Related Topics

[\\$cwb show](#)

\$cwb viewedLayoutClbk

Changes the currently-displayed layout to a specified layout object.

Usage

\$cwb viewedLayoutClbk *layout_handle*

Arguments

- *layout_handle*

A required argument specifying the new layout to display.

Return Values

None.

Description

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

Examples

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

\$cwb waitForDrawing

Measures the time it takes for the last-issued drawing command to complete.

Usage

\$cwb waitForDrawing [*timeout*]

Arguments

- *timeout*

An optional argument specifying the maximum wait time in seconds, not longer than 120 seconds (2 minutes).

Return Values

Returns a time in microseconds.

Description

Measures the time it takes for the last-issued drawing command to complete. The command measures time in microseconds.

Note



The command begins measuring time from the moment the command is called. This functionality is useful in a script just after the drawing command is issued.

Examples

```
% cat script.tcl
set cwb ::cwbWorkBench::cwbWorkBench0
$cwb zoomToRectClbk 10 10 5000 5000
set dt [$cwb waitForDrawing]
puts "Update in $dt us"

% calibredrv -s script.tcl common.gds
```

Related Topics

[\\$cwb zoomToRectClbk](#)

\$cwb zoomAllClbk

Fills the display with the entire layout design.

Usage

\$cwb zoomAllClbk

Arguments

None.

Return Values

None.

Description

Fills the display with the entire current cell, zooming as needed.

Examples

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

Related Topics

[\\$cwb zoomToRectClbk](#)

\$cwb zoomToRectClbk

Zooms the window to the specified coordinates.

Usage

\$cwb zoomToRectClbk *llx lly urx ury*

Arguments

- *llx lly urx ury*

A required set of four arguments specifying the lower left and upper right coordinates to zoom to in the layout. The coordinates are in database units (dbu).

Return Values

None.

Description

Zooms to the specified rectangular region in the layout viewer.

Examples

```
set wb [find objects -class cwbWorkBench]
$cwb zoomToRectClbk 100 100 4900 4900
```

Related Topics

[\\$cwb zoomAllClbk](#)

Layout Commands

The layout commands are used to manipulate entire layout objects, including operations such as creating, deleting, and merging layouts.

Table 5-6. Layout Commands Summary

Layout Commands	Description
layout all	Returns a list of all current layout object handles.
layout copy	Copies objects from a source layout into a destination layout, after flattening it to the specified depth.
layout copy2	Copies all geometries and cell references that touch or are contained within the selection region, preserving any hierarchy that exists.
layout create Commands	The layout create commands are used to create a new layout, or to create a view of or manipulate an existing layout. Different usages of the layout create command are available to support different layout database formats or methods for creating the layout view.
layout createCache	Creates a cache file that can be used to enable faster loading of a layout.
layout delete	Deletes the specified layout.
layout droasis	Opens a disk-resident OASIS file in read-only mode.
layout filemerge	Performs a disk-based merge of multiple GDS files, multiple OASIS files, or a combination of GDS and OASIS files without loading them into memory.
layout filtershapes	Removes polygon shapes from the layout file hierarchy.
layout merge	Merges two layouts, with automatic renaming of the cells to prevent name conflicts.
layout overlays	Returns all defined overlay handles.
layout peek	Returns information from a layout database without loading the file into memory.
layout tbmode	Specifies the text box extent treatment behavior.

layout all

Returns a list of all current layout object handles.

Usage

layout all

Arguments

None.

Return Values

A list of all layout object handles.

Description

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

Examples

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

Related Topics

[overlay all](#)

layout copy

Copies objects from a source layout into a destination layout, after flattening it to the specified depth.

Note



This command is not supported in HC mode.

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 to copy the data. Allowed values are a user-supplied string, which cannot currently be in use as the handle of an existing layout.
If not specified, the tool generates a layout handle for the new layout.
- *cell*
An optional argument specifying the name of the cell in the source layout to copy. The cell becomes the topcell in the new layout. To copy the entire layout, specify the name of the topcell.
- *startDepth endDepth*
An optional argument set specifying the hierarchy depth range containing the data to be copied. Before copying, the tool flattens the data between the *startDepth* and the *endDepth*. Any data above *startDepth* or below *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 set specifying the coordinates of a rectangular selection area. The first two values define the lower left corner while the second two values define the upper right corner. If not specified, the entire layout is copied. By default, coordinates are in database units (dbu).
Coordinates are 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 argument that specifies the selection mode to apply to the selection area. Allowed values are 0, 1, or 2. Alternatively, values can be “contain”, “touch”, or “clip”.

- 0 (contain) — Copies all objects contained within the selection area. This is the default.
- 1 (touch) — Copies all objects that touch or are contained within the selection area.
- 2 (clip) — Copies all objects contained within the selection area. For objects whose shapes are not entirely contained within the selection area, clip mode specifies to copy only the part of the shape inside the selection area.

- -preserveTextAttributes

An optional argument instructing the tool to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the tool. Use this argument when the source layout is a GDS layout. The default is to not preserve these attributes.

- -preserveProperties

An optional argument instructing the tool to preserve GDS geometry or reference property data (PROPATTR and PROPVALUE) when reading data into or writing data from the tool. Use this argument when copying a layout from a GDS layout. The default is to not preserve these attributes.

- -preserveHierarchy

An optional argument specifying to preserve SREF- and AREF-instances that are in the selection area. The default is to not preserve the hierarchy, creating a flat layout copy.


Return Values

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

Description

Flattens data and copies it from a source layout into a destination layout. The command provides arguments, such as *cell* and *startDepth/endDepth*, that allow you to define which data in the specified layout to copy.

Note

 Most of the arguments for this method are order-dependent. The way the tool recognizes what an argument signifies is by assessing its position in the command string. Due to this, you cannot specify an argument toward the end of the command string without specifying all arguments that precede it. Thus, while most arguments are technically optional, they are necessary in many situations.

Examples

Example 1

This example copies the entire layout to a new flattened version of the original layout:

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

Example 2

This example copies a portion of the cell named “mycell” 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 mycell 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 mycell $layer range 0 end]] {
        puts "layer($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
```

Example 3

This example copies polygons in a given coordinate window, excluding parts of shapes not contained within it:

```
% basename [info nameofexecutable]
calibrewb
% set layout [layout create original.gds]
% layout copy layout0 newlayout topcellname 0 99 $x1 $y1 $x2 $y2 2 \
    -preserveHierarchy
% set L newlayout
% $L gdsout output.gds -log output.log
```

Related Topics

[layout copy2](#)

layout copy2

Copies all geometries and cell references that touch or are contained within the selection region, preserving any hierarchy that exists.

Note



This command is not supported in HC mode.

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 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.
- **x1 y1 x2 y2**
A required argument set specifying the coordinates of the rectangular selection area. The first two values define the lower left corner, the second two define the upper right corner. By default, coordinates are in database units (dbu).
Coordinates are 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 argument instructing the tool to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the tool. Use this argument when copying a layout from a GDS layout. The default is to not preserve these attributes.
- **-preserveProperties**
An optional argument instructing the tool to preserve GDS geometry or reference property data (PROPATTR and PROPVALUE). Use this argument when copying a layout from a GDS file. The default is to not preserve these properties.

Return Values

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

Description

Copies all geometries and cell references that touch or are contained within the selection region to a new layout object, preserving any hierarchy that exists. Cell references do not contain geometric data.

Examples

Example 1

This example copies data from layout2 to a new layout, layout5:

```
% layout all
layout2 layout3 layout4
% layout copy2 layout2 a1220 0 0 200000 200000
layout5
```

Example 2

This example copies a portion of the cell named “mycell” to a new layout, and sorts through the data:

```
% set L_copy2 [layout copy2 layout0 mycell -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 mycell $layer range 0 end]] {
        puts "layer($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
```

Related Topics

[layout copy](#)

layout create Commands

The layout create commands are used to create a new layout, or to create a view of or manipulate an existing layout. Different usages of the layout create command are available to support different layout database formats or methods for creating the layout view.

Table 5-7. layout create Commands Summary

layout create Commands	Description
layout create	Creates a new layout.
layout create (GDS or OASIS file)	Creates a view of a layout from an existing GDS or OASIS® layout database.
layout create (ASCII RDB)	Creates a view of a layout from an existing ASCII DRC Results Database (RDB) file.
layout create (LEF/DEF or OpenAccess)	Creates a view of a layout from an existing LEF/DEF or OpenAccess layout database.
layout create (mapped layout)	Creates a view of a layout from an existing GDS or OASIS file, with the ability to map layers or layers and datatypes.
layout create (multiple layouts)	Creates a view of a layout from multiple existing layout databases.

layout create

Creates a new layout.

Usage

layout create [-type {gds | oasis}] [-handle *handle*]

Arguments

- -type {gds | oasis}
An optional argument specifying the database file format for the layout. The default format when this argument is not specified is GDS.
- -handle *handle*
An optional argument specifying a handle name to assign to the newly-created layout handle.

Return Values

The handle for the new layout object. The handle that is returned is either a system-generated value or the handle that is specified when executing the command.

Description

This form of the layout create command creates an empty in-memory layout. It does not display the layout until you add a cell or layer. This command is useful for creating a new empty layout and subsequently adding cells and other objects.

Examples

Example 1

This example creates a new empty GDS layout and assigns it the handle “new_layout”. It uses the handle to create a cell, a layer, and a polygon. The results are output to the file *my_layout.gds*.

```
layout create -type gds -handle new_layout
new_layout create cell top
new_layout create layer 20
new_layout create polygon top 20 500 500 2000 2000
new_layout gdsout my_layout.gds
```


layout create (GDS or OASIS file)

Creates a view of a layout from an existing GDS or OASIS^{®1} layout database.

Note



The -pcr_file, -validate_pcr, and -log arguments are not supported in HC mode.

Usage

```
layout create [handle] fileName [-dt_expand] [-preservePaths] [-ignoreGdsBoxes]
[-preserveTextAttributes] [-preserveProperties] [-handle handle] [-ignoreInsts]
[-ignorePaths] [-ignorePolys] [-ignoreTexts] [-incr [sync]] [-pcr_file pathname]
[-validate_pcr level] [-log logfile]
```

Note



The most commonly used arguments when creating a layout view from a GDS or OASIS file are -dt_expand, -preservePaths, -preserveTextAttr (GDS only), and -preserveProperties.

Arguments

- *handle*
An optional argument specifying a handle name to assign to the new layout. If *handle* is not specified, the tool assigns the new layout a system-generated handle. The format for a system-generated handle is “layout*N*”. The system-generated handle assigned to the first layout loaded into memory is layout0.
- *fileName*
A required argument specifying the name of a GDS or OASIS layout database file to use when creating the layout view.
- -dt_expand
An optional argument that expands datatypes so that each layer and datatype combination is mapped to a different layer in the new layout. The format used for each layer is *layer.datatype* (for example, 12.134). When this argument is not specified, layers with the same layer number but different datatypes are merged into a single layer with the same layer number.
- -preservePaths
An optional argument that preserves path definitions when reading an input layout database. By default, positive-width paths are converted to polygons. Zero-width paths are not expanded.
- -ignoreGdsBoxes
An optional argument that instructs the tool to ignore box records when reading a GDS file.

1. OASIS[®] is a registered trademark of Thomas Grebinski and licensed for use to SEMI[®], San Jose. SEMI[®] is a registered trademark of Semiconductor Equipment and Materials International.

- **-preserveTextAttributes**

An optional argument that preserves GDS Text Presentation and Strans attribute data when reading an input layout database. The default is to not preserve text attributes. Preserving text attributes does not affect the loading of text objects, as text objects are always loaded. This argument is ignored with OASIS files because the OASIS file format does not support presentation attributes in text records.

- **-preserveProperties**

An optional argument that preserves any property data when reading an input layout database.

- **-handle *handle***

An optional argument specifying a handle name to assign to the newly-created layout handle. This argument is ignored if a value is specified for the *handle* argument.

- **-ignoreInsts**

An optional argument used to disable the loading of instances from the specified layout database. Any child cells of instances are also ignored.

- **-ignorePaths**

An optional argument used to disable the loading of paths from the specified layout database.

- **-ignorePolys**

An optional argument used to disable the loading of polygons from the specified layout database.

- **-ignoreTexts**

An optional argument used to disable the loading of texts from the specified layout database.

- **-incr [sync]**

An optional argument that invokes the layout viewer in incremental mode, using a Peek Cache Repository (PCR) file to aid the initial drawing speed. The tool generates or updates a PCR file as needed. Only hierarchical structures are loaded when a layout is opened in incremental mode.

The sync option forces the syncing of the PCR file with the layout.


For more information on the PCR file, refer to “[The Peek Cache Repository File](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual*.

- **-pcr_file *pathname***

An optional argument used to specify the path and filename of a PCR file. Use this switch with the -incr argument to specify a PCR file that aids the initial drawing speed when loading a layout in incremental mode. This argument is only supported when loading a single layout database and issues an error message if used when loading multiple layout databases.

The `-pcr_file` argument overrides the `MGC_CWB_PCR_PATH` environment variable. The `MGC_CWB_PCR_PATH` environment variable specifies a directory location for the PCR files, while the `-pcr_file` argument specifies the path to a PCR file.

Note


 This argument is not supported in HC mode.

- `-validate_pcr level`

An optional argument used to perform checks to determine if a PCR file is usable prior to loading the layout. Valid values include:

- 0 — No check is performed (default).
- 1 — The PCR is checked to make sure it is valid for the layout. If the PCR is valid, then the layout is loaded. If it is not valid, the layout create command will fail with an error message stating what is wrong with the PCR file.
- 2 — The PCR is checked to make sure it is valid for the layout, but the layout is not loaded regardless of whether the PCR is valid or not. This option generates a note if the PCR file is usable and an error message if the PCR file is not usable.


Note

 This argument is not supported in HC mode.

- `-log logfile`

An optional argument specifying the name of a log file in which to output information from the layout create operation. The log file includes the transcript and details (such as cell, layer, and object summaries) about the layout being created. Specify “stdout” for *logfile*, to print the transcript to the terminal window.

Note

 This argument is not supported in HC mode.

Return Values

The handle assigned to the new layout view. The handle that is returned is either a system-generated value or the handle that is specified when executing the command.

Description

Creates a view of a layout from a GDS or OASIS file. To create an empty in-memory layout, refer to the [layout create](#) command.

The specified *fileName* may be a compressed file, but must have a suffix of `.z`, `.Z`, or `.gz`, and the `gzip` program must be accessible to read the compressed file.

Examples

Example 1

This example creates a view of *mylayout.gds*. The `-dt_expand` argument ensures each layer and datatype combination is mapped to a different layer in the new layout. The `-preserveProperties` argument preserves any geometry and reference property data.

```
set L [layout create mylayout.gds -dt_expand -preserveProperties]
puts [$L layers]
$L gdsout ./output/new_layout.gds
```

The `$L layers` command returns a list of the layer numbers in the layout, and `$L gdsout` writes the results to *new_layout.gds*.

Example 2

This example creates a view of *mix.gds* and assigns the results to the variable `L`. The `-preserveTextAttributes` argument preserves GDS Text Presentation and Strans attribute data, while the `-preservePaths` argument preserves path definitions.

```
set L [layout create mix.gds -preserveTextAttributes -preservePaths]
puts [$L layers]
$L gdsout ./output/new_layout.gds
```

The `$L layers` command returns a list of the layer numbers in the layout, and `$L gdsout` writes the results to *new_layout.gds*:

```
4 7 1 2 3 5 11
Writing GDSII file: "./output/new_layout.gds"
```

Example 3

This example creates a view of *layout.oas* and assigns it the handle “myhandle.” The `-dt_expand` argument expands datatypes, mapping each layer and datatype combination to a different layer in the new layout.

```
layout create layout.oas -dt_expand -handle myhandle
puts [myhandle cells]
myhandle oasisout ./output/new_layout.oas
```

The user-defined handle (`myhandle`) returns a list of cells in the layout and then outputs the layout database to an OASIS file:

```
CB14_2A_BLOCK1
Writing file: "./output/new_layout.oas"
```

layout create (ASCII RDB)

Creates a view of a layout from an existing ASCII DRC Results Database (RDB) file.

Usage

```
layout create [handle] fileName [-layer lindex] [-rdbChecks checknames]  
[-singleLayer] [-preserveProperties] [-handle handle] [-ctoFile filename]
```

Note



The most commonly used argument when creating a layout view from an ASCII RDB is -preserveProperties.

Arguments

- *handle*
An optional argument specifying a handle name to assign to the new layout. If *handle* is not specified, the tool assigns the new layout a system-generated handle. The format for a system-generated handle is “layout*N*”. The system-generated handle assigned to the first layout loaded into memory is layout0.
- **fileName**
A required argument specifying the name of an ASCII DRC Results Database (RDB) file to use when creating the layout view.
- -layer *lindex*
An optional argument that specifies a starting layer number to use when creating the layout. Additional layers are incremented from the *lindex* starting layer.
- -rdbChecks *checknames*
An optional argument specifying the name of one or more RDB checks to use when creating the layout view. Check names are case-sensitive and wildcards (*) can be used to find zero or more occurrences of a check name. For example, “ch*” finds ch1, check2, and chk3b. All checks are loaded by default.
- -singleLayer
An optional argument that specifies to use a single layer to represent multiple checks. By default, each check is placed on its own layer and the check names are used for the layernames.
- -preserveProperties
An optional argument that preserves property data when reading an input results database.
- -handle *handle*
An optional argument specifying a handle name to assign to the newly-created layout handle. This argument is ignored if a value is specified for the *handle* argument.

- `-ctoFile filename`

An optional argument used to create a Check Text Override (CTO) file containing the names of the checks and corresponding check text that exists in the specified RDB file. You can edit the resulting CTO file to include DRC RVE rule check comments and you can load a CTO file into a Calibre RVE for DRC session.

For information on CTO Files, refer to “[DRC RVE Check Text Override File \(CTO File\)](#)” in the *Calibre RVE User’s Manual*.

Return Values

The handle for the new layout object. The handle that is returned is either a system-generated value or the handle that is specified when executing the command.

Description

Creates a layout view from an ASCII DRC RDB file. To create an empty layout, refer to the [layout create](#) reference page.

When loading an ASCII DRC RDB file, Calibre DESIGNrev automatically segments polygons that exceed 8192 vertices. Edge clusters are converted to wire (or path) object types. Calibre DESIGNrev segments any edge clusters having more than 4096 edges (8192 vertices).

Examples

Example 1

This example creates a layout view from the results in the RDB file named *fullchip.rdb* and assigns the new layout the handle “fullchip”. It then outputs the results to a GDS file named *new_fullchip.gds*.

```
set L [layout create fullchip fullchip.rdb]
$L gdsout ./output/new_fullchip.gds
```

Example 2

This example creates a layout view from the results in the RDB file named *fullchip.rdb*, with layer numbers starting at 100. The `$L gdsout` command outputs the results to a file named *new_fullchip.gds*.

```
set L [layout create fullchip fullchip.rdb -layer 100]
$L gdsout ./output/new_fullchip.gds
```

Example 3

This example creates a layout from the results in the RDB file named *fullchip.rdb*. The `-rdbChecks` argument specifies to use the results from the checks named “NW.2” and “OX.3” to create the layout. The `$L gdsout` command outputs the results to a file named *new_fullchip.gds*.

```
set L [layout create fullchip.rdb -rdbChecks NW.2 OX.3]
$L gdsout ./output/new_fullchip.gds
```

Example 4

This example creates a layout from the results in the RDB file named *drc.results* and creates a CTO file named *drc.results.cto*. The resulting output file, named *my_drc_results.rdb*, includes only the cell named “lab4a” and layers 1 through 6.

```
set L [layout create drc.results -ctoFile ./output/drc.results.cto]  
$L rdbout ./output/my_drc_results.rdb -cell lab4a -layers 1 2 3 4 5 6
```

layout create (LEF/DEF or OpenAccess)

Creates a view of a layout from an existing LEF/DEF or OpenAccess layout database.

Usage

layout create

```
{ {LEFDEF -def {def_file | @def_empty} -lef lef_files} | {OA library cell view} }  
[-dt_expand] [-preservePaths] [-preserveProperties] [‘{-lefdefOptions options...’}]
```

Arguments

- **LEFDEF -def {def_file | @def_empty} -lef lef_files**

A required argument set when creating a layout from LEF/DEF database files. Valid options include:

-def {def_file | @def_empty} — Specifies the name of the DEF file to use when creating the layout. Use **@def_empty** if there is no DEF file.

-lef lef_files — Specifies the names of the LEF files to use when creating the layout. The LEF files should be specified using the Tcl list command and be enclosed in brackets, with the LEF technology file occurring first in the list of files.

- **OA library cell view**

A required argument set when creating a layout from an OpenAccess (OA) database. The **library cell view** options specify the name of the OA library, cell, and view to use when creating the layout. OpenAccess must be installed to access the OA database and read the data.

- **-dt_expand**

An optional argument that expands datatypes so that each layer and datatype combination is mapped to a different layer in the new layout. The format used for each layer is *layer.datatype* (for example, 12.134). When this argument is not specified, layers with the same layer number but different datatypes are merged into a single layer with the same layer number.

- **-preservePaths**

An optional argument that preserves path definitions when reading an input layout database. By default, positive-width paths are converted to polygons. Zero-width paths are not expanded.

- **-preserveProperties**

An optional argument that preserves any geometry and reference property data (PROPATTR and PROPVALUE) when reading an input layout database.

- **‘{-lefdefOptions options...’}**

An optional argument used to specify options for modifying the default read-in behavior of the LEF/DEF database. Multiple options can be specified in this argument. You can set the

CWB_ECHO_FDI_CMD environment variable to any value to enable transcribing of the arguments passed to the fdi2oasis command.

Refer to “[Complete FDI Command Line Syntax](#)” in the *Calibre Layout Comparison and Translation Guide* for descriptions of the supported options. All options described in the *Calibre Layout Comparison and Translation Guide* are supported except for the following:

- **-design library cell** [view]
- -abortOnEmptyPCell
- -libdefs *filename* ...
- -preservePath

Return Values

The system-generated handle assigned to the new layout view.

Description

Creates a view of a layout from an existing LEF/DEF or OA layout database.

Examples

Example 1

This example creates a LEF/DEF layout database from the specified LEF and DEF files. Options specify how pins, nets, and instances are annotated. The results for layers 0 and 5 in the top cell and are output to *layout.rdb*.

```
layout create LEFDEF -def ./DEF/top.def \  
-lef [list ./LEF/tech.lef ./LEF/ADC_5bit_sc_5.lef ./LEF/dac6_op2.lef \  
./LEF/gscl45nm.lef ./LEF/myram.lef ./LEF/PADS.lef] \  
{-lefdefOptions -annotatePins TEXT -annotateNets TEXT -annotateInsts 6}  
$L rdbout ./output/layout.rdb -cell top -layers 0 5
```

For this example, setting the CWB_ECHO_FDI_CMD environment variable to any value transcribes the information that is passed to the fdi2oasis command as shown here:

```
Invoking FDI with: <path>/fdi2oasis -64 -system LEFDEF -def ./DEF/top.def  
-lef ./LEF/tech.lef ./LEF/ADC_5bit_sc_5.lef ./LEF/dac6_op2.lef ./LEF/  
gscl45nm.lef ./LEF/myram.lef ./LEF/PADS.lef -annotatePins TEXT  
-annotateNets TEXT -annotateInsts 6
```

layout create (mapped layout)

Creates a view of a layout from an existing GDS or OASIS file, with the ability to map layers or layers and datatypes.

Note



The `-pcr_file`, `-validate_pcr`, and `-log` arguments are not supported in HC mode.

Usage

layout create [*handle*] *fileName*

```
[ { [ {-mapname Lname [LMAP]} ... ] [ {-map L D LMAP [DMAP]} ... ] [-only] }  
  | {-mapfile map_file [-only]} ]  
[-dt_expand] [-preservePaths] [-ignoreGdsBoxes] [-preserveTextAttributes]  
[-preserveProperties] [-noReport] [-incr [sync]] [-pcr_file pathname] [-validate_pcr level]  
[-handle handle] [-levels level1 ... leveln] [-log logfile]
```

Arguments

- *handle*

An optional argument specifying a handle name to assign to the new layout. If *handle* is not specified, the tool assigns the new layout a system-generated handle. The format for a system-generated handle is “layout*N*”. The system-generated handle assigned to the first layout loaded into memory is layout0.

- *fileName*

A required argument specifying the name of a GDS or OASIS layout database file to use when creating the layout view.

- {-mapname *Lname* [*LMAP*]} ... [-only]

An optional argument set used to map a layername in a GDS or OASIS file to a layer number in the new layout. You can specify the `-mapname` argument multiple times to define more than one layername and layer number mapping. The `-mapname` and `-map` arguments can also be specified together to define more than one mapping. Valid options include:

Lname — The name of a layer in the GDS or OASIS file.

LMAP — An optional integer value specifying the layer number in the new layout to which the GDS or OASIS layer is to be mapped. If *LMAP* is omitted, the layout object uses the layer number assigned to the *Lname* layer.

`-only` — An optional argument that specifies to load only the layers explicitly defined by the `-mapname` argument.

If `-only` is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

Any layers that are not mapped using the `-mapname` argument are loaded and assigned the same layer number as in the GDS or OASIS file.

- {-map *L D LMAP [DMAP]... [-only]*

An optional argument used to map a layer and datatype pair in a GDS or OASIS file to a layer in the new layout. The *L*, *D*, and *LMAP* values are required when -map is specified. You can specify the -map argument multiple times to define more than one mapping. The -map and -mapname arguments can also be specified together to define more than one mapping. Valid options include:

L D — A set of values specifying the layer and datatype in the GDS or OASIS file. A value of *D* < 0 (for example, -1) matches any datatype.

LMAP — An integer value specifying the layer number in the new layout to which the GDS or OASIS layer and datatype pair are to be mapped.

DMAP — An optional integer value specifying a datatype associated with *LMAP*. If *DMAP* is not specified, a datatype value of 0 is assumed.

-only — An optional argument that specifies to load only the layers explicitly defined by the -map argument.

If -only is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

- -mapfile *map_file* [-only]

An optional argument used to provide a layer map file for GDS or OASIS design input.

map_file — The layer map file, which is a text file with this syntax:

L D LMAP [.DMAP]

where:

L D — A set of values specifying the layer and datatype in the GDS or OASIS file. A value of *D* < 0 (for example, -1) matches any datatype.

LMAP — An integer value specifying the layer number in the new layout to which the GDS or OASIS layer and datatype pair are to be mapped.

DMAP — An optional integer value specifying a datatype associated with *LMAP*. If *DMAP* is not specified, a datatype value of 0 is assumed.

Each mapped layer is listed on a separate line. See the description section for more details.

All layers are loaded unless -only is specified. Any layers not listed in *map_file* are assigned the same layer number as in the GDS or OASIS file.

-only — An optional argument that specifies to load only the layers explicitly defined in the *map_file*. If -only is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

The -mapfile argument set cannot be used with -mapname or -map.

- **-dt_expand**
An optional argument that expands datatypes so that each layer and datatype combination is mapped to a different layer in the new layout. The format used for each layer is *layer.datatype* (for example, 12.134). When this argument is not specified, layers with the same layer number but different datatypes are merged into a single layer with the same layer number.
- **-preservePaths**
An optional argument that preserves path definitions when reading an input layout database. By default, positive-width paths are converted to polygons. Zero-width paths are not expanded.
- **-ignoreGdsBoxes**
An optional argument that instructs the tool to ignore box records when reading a GDS file.
- **-preserveTextAttributes**
An optional argument that preserves GDS Text Presentation and Strans attribute data when reading an input layout database. The default is to not preserve text attributes. Preserving text attributes does not affect the loading of text objects, as text objects are always loaded. This argument is ignored with OASIS files because the OASIS file format does not support presentation attributes in text records.
- **-preserveProperties**
An optional argument that preserves any geometry and reference property data (PROPATTR and PROPVALUE) when reading an input layout database.
- **-noReport**
An optional argument that 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.
- **-incr [sync]**
An optional argument that invokes the layout viewer in incremental mode, using a Peek Cache Repository (PCR) file to aid the initial drawing speed. The tool generates or updates a PCR file as needed. Only hierarchical structures are loaded when a layout is opened in incremental mode.


The sync option forces the syncing of the PCR file with the layout.

For more information on the PCR file, refer to “[The Peek Cache Repository File](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual*.
- **-pcr_file pathname**
An optional argument used to specify the path and filename of a PCR file. Use this switch with the -incr argument to specify a PCR file that aids the initial drawing speed when loading a layout in incremental mode. This argument is only supported when loading a

single layout database and issues an error message if used when loading multiple layout databases.

The `-pcr_file` argument overrides the `MGC_CWB_PCR_PATH` environment variable. The `MGC_CWB_PCR_PATH` environment variable specifies a directory location for the PCR files, while the `-pcr_file` argument specifies the path to a PCR file.

Note


 This argument is not supported in HC mode.

- `-validate_pcr level`

An optional argument used to perform checks to determine if a PCR file is usable prior to loading the layout. Valid values include:

- 0 — No check is performed (default).
- 1 — The PCR is checked to make sure it is valid for the layout. If the PCR is valid, then the layout is loaded. If it is not valid, the layout create command will fail with an error message stating what is wrong with the PCR file.
- 2 — The PCR is checked to make sure it is valid for the layout, but the layout is not loaded regardless of whether the PCR is valid or not. This option generates a note if the PCR file is usable and an error message if the PCR file is not usable.

Note

 This argument is not supported in HC mode.

- `-handle handle`

An optional argument specifying a handle name to assign to the newly-created layout handle. This argument is ignored if a value is specified for the *handle* argument.


- `-levels level1 ... leveln`

An optional argument used only in Calibre MDPview. Specifies a list of levels used for normal, extended, and flex MEBES job decks. Using a level list reduces load times and memory usage for job decks that have assigned unique files to each level, since only those pattern or index files that match a level in the list are loaded. Using a level list on the other job deck types (Hitachi, JEOL, VSB, Micronic) has no effect.

- `-log logfile`

An optional argument specifying the name of a log file in which to output information from the layout create operation. The log file includes the transcript and details (such as cell, layer, and object summaries) about the layout being created. Specify “stdout” for *logfile*, to print the transcript to the terminal window.

Note

 This argument is not supported in HC mode.

Return Values

The handle for the new layout object. The handle that is returned is either a system-generated value or the handle that is specified when executing the command.

Description

Creates a view of a layout from an existing GDS or OASIS file, with the ability to map layers or layers and datatypes.

When using the `-mapname`, `-map`, or `-mapfile` arguments, see “[Using Layer Maps](#)” in the Calibre DESIGNrev Layout Viewer User’s Manual for general information.

Examples

Example 1

The example reads in only the layername `ntub` and then outputs the results to `my_layout.oas`.

```
set L [layout create fullchip.oas -mapname ntub -only]
$L oasisout ./output/my_layout.oas
```

Example 2

This example reads in all layers, but maps the layernames `ntub` to 300, `nimplant` to 400, and `pimplant` to 500. It then outputs the results to `my_layout.oas`.

```
set L [layout create fullchip.oas -mapname ntub 300 \
      -mapname nimplant 400 -mapname pimplant 500]
$L oasisout ./output/my_layout.oas
```

Example 3

This example creates a new layout view by mapping layers 1045 to layer 222. The `-l` value specifies to match any datatype value that is on layer 1045. The results are output to `my_layout.oas`

```
set L [layout create layout.oas -map 1045 -l 222]
$L oasisout ./output/my_layout.oas
```

Example 4


This example creates a new layout view by mapping layer 1 with any datatype value to layer 10 and mapping layer 2 with any datatype value to layer 20. The `-dt_expand` argument maps each layer and datatype combination to a different layer. The `$L layernames` command writes the details of the layers in the OASIS layout to the file `layers.txt`. The results of the mapped layout are output to `my_layout.oas`.

```
set L [layout create layout.oas -map 1 -l 10 -map 2 -l 20 \
      -dt_expand]
set outfile [open "./output/layers.txt" w]
puts $outfile [$L layernames -oasis]
close $outfile
$L oasisout ./output/my_layout.oas
```

layout create (multiple layouts)

Creates a view of a layout from multiple existing layout databases.

Note

 The -log argument is not supported in HC mode.

Usage

```
layout create [handle] {-files fileNameList}  
  [{ {-mapname Lname [LMAP]} ...} [{-map L D LMAP [DMAP]} ...} [-only] }  
  | {-mapfile map_file [-only]} ]  
  [-dt_expand] [-preservePaths] [-ignoreGdsBoxes]  
  [-preserveTextAttributes] [-preserveProperties] [-noReport] [-handle handle] [-log logfile]
```

Arguments

- *handle*
An optional argument specifying a handle name to assign to the new layout. If *handle* is not specified, the tool assigns the new layout a system-generated handle. The format for a system-generated handle is “layout*N*”. The system-generated handle assigned to the first layout loaded into memory is layout0.
- -files *fileNameList*
A required argument used to specify a list of filenames for the layout databases that are to be concatenated together to create the new layout view. Each filename specified in the list must be separated by a space.

When you specify multiple input files in which the input precisions do not match, the lowest common multiple of the input precisions is calculated. Each input layout is then magnified as it is read to ensure the user unit values are retained.
- {-mapname *Lname* [*LMAP*]} ... [-only]
An optional argument set used to map a layername in a GDS or OASIS file to a layer number in the new layout. You can specify the -mapname argument multiple times to define more than one layername and layer number mapping. The -mapname and -map arguments can also be specified together to define more than one mapping. Valid options include:
 - Lname* — The name of a layer in the GDS or OASIS file.
 - LMAP* — An optional integer value specifying the layer number in the new layout to which the GDS or OASIS layer is to be mapped. If *LMAP* is omitted, the layout object uses the layer number assigned to the *Lname* layer.
 - only — An optional argument that specifies to load only the layers explicitly defined by the -mapname argument.

If -only is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

Any layers that are not mapped using the `-mapname` argument are loaded and assigned the same layer number as in the GDS or OASIS file.

- `{-map L D LMAP [DMAP] ... [-only]`

An optional argument used to map a layer and datatype pair in a GDS or OASIS file to a layer in the new layout. The *L*, *D*, and *LMAP* values are required when `-map` is specified. You can specify the `-map` argument multiple times to define more than one mapping. The `-map` and `-mapname` arguments can also be specified together to define more than one mapping. Valid options include:

L D — A set of values specifying the layer and datatype in the GDS or OASIS file. A value of *D* < 0 (for example, -1) matches any datatype.

LMAP — An integer value specifying the layer number in the new layout to which the GDS or OASIS layer and datatype pair are to be mapped.

DMAP — An optional integer value specifying a datatype associated with *LMAP*. If *DMAP* is not specified, a datatype value of 0 is assumed.

`-only` — An optional argument that specifies to load only the layers explicitly defined by the `-map` argument.

If `-only` is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

- `-mapfile map_file [-only]`

An optional argument used to provide a layer map file for GDS or OASIS design input.

map_file — The layer map file, which is a text file with this syntax:

L D LMAP [.DMAP]

where:

L D — A set of values specifying the layer and datatype in the GDS or OASIS file. A value of *D* < 0 (for example, -1) matches any datatype.

LMAP — An integer value specifying the layer number in the new layout to which the GDS or OASIS layer and datatype pair are to be mapped.

DMAP — An optional integer value specifying a datatype associated with *LMAP*. If *DMAP* is not specified, a datatype value of 0 is assumed.

Each mapped layer is listed on a separate line. See the description section for more details.

All layers are loaded unless `-only` is specified. Any layers not listed in *map_file* are assigned the same layer number as in the GDS or OASIS file.

`-only` — An optional argument that specifies to load only the layers explicitly defined in the *map_file*. If `-only` is not specified, then all layers are loaded, including those layers that are not explicitly mapped. The unmapped layers are assigned the same layer number as in the GDS or OASIS file.

The `-mapfile` argument set cannot be used with `-mapname` or `-map`.

- **-dt_expand**
An optional argument that expands datatypes so that each layer and datatype combination is mapped to a different layer in the new layout. The format used for each layer is *layer.datatype* (for example, 12.134). When this argument is not specified, layers with the same layer number but different datatypes are merged into a single layer with the same layer number.
- **-preservePaths**
An optional argument that preserves path definitions when reading an input layout database. By default, positive-width paths are converted to polygons. Zero-width paths are not expanded.
- **-ignoreGdsBoxes**
An optional argument that instructs the tool to ignore box records when reading a GDS file.
- **-preserveTextAttributes**
An optional argument that preserves GDS Text Presentation and Strans attribute data when reading an input layout database. The default is to not preserve text attributes. Preserving text attributes does not affect the loading of text objects, as text objects are always loaded. This argument is ignored with OASIS files because the OASIS file format does not support presentation attributes in text records.
- **-preserveProperties**
An optional argument that preserves any geometry and reference property data (PROPATTR and PROPVALUE) when reading an input layout database.
- **-noReport**
An optional argument that 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.
- **-handle *handle***
An optional argument specifying a handle name to assign to the newly-created layout handle. This argument is ignored if a value is specified for the *handle* argument.
- **-log *logfile***
An optional argument specifying the name of a log file in which to output information from the layout create operation. The log file includes the transcript and details (such as cell, layer, and object summaries) about the layout being created. Specify “stdout” for *logfile*, to print the transcript to the terminal window.

Note

This argument is not supported in HC mode.

Return Values

The handle for the new layout object. The handle that is returned is either a system-generated value or the handle that is specified when executing the command.

Description

Creates a view of a layout by concatenating multiple layout files. To merge multiple layout files, use the [layout filemerge](#) command.

When using the `-mapname`, `-map`, or `-mapfile` arguments, see “[Using Layer Maps](#)” in the Calibre DESIGNrev Layout Viewer User’s Manual for general information.

Examples

Example 1

This example creates a layout view from the layouts *spare_top_fill.oas* and *spare_top_wIP.oas*, and assigns the layout view the handle “fill_wIP”. The `-dt_expand` argument ensures each layer and datatype combination in the input layouts is mapped to different layers in the new layout. The results are then written to *spare_top_fill_wIP.oas*.

```
set L1 [layout create -files spare_top_fill.oas spare_top_wIP.oas \  
-handle fill_wIP -dt_expand]  
$L1 oasisout ./output/spare_top_fill_wIP.oas
```

Example 2


This example creates a layout view from *mix.oas* and *lab4a.oas*. Layer 1, datatype 0 is mapped to layer 100, datatype 0 and layer 5, datatype 0 is mapped to layer 500, datatype 0. These are the only layers loaded in the new layout. The results are then written to *lab4a_mix.oas*.

```
set L1 [layout create -files mix.oas lab4a.oas -map 1 0 100 0  
-map 5 0 500 0 -only]  
$L1 oasisout ./output/lab4a_mix.oas
```

layout createCache

Creates a cache file that can be used to enable faster loading of a layout.

Note

 This command is only supported in HC mode.

Usage

layout createCache *path* [*cachePath*]

Arguments

- *path*
A required argument specifying the path to the input layout to use for generating the cache file. There is no restriction on the format of the input layout format.
- *cachePath*
An optional argument specifying the name of the output cache file. If *cachePath* is not specified, the cache file is output to the current working directory (if writable) using the layout name and appending a .clv suffix.

Description

Creates a cache file for the input layout specified by *path*.

You can create a cache file for any type of layout file that is supported in HC mode, including gzip compressed and GDS files.

Examples

This example creates a cache file for *layout.oas* and saves the output to *layout.oas.clv*.

```
% layout createCache layout.oas <path>/layout.oas.clv
```

layout delete

Deletes the specified layout.

Usage

layout delete *handle*

Arguments

- *handle*

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

Return Values

None.

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 \$scwb deleteLayoutClbk command to avoid generating errors.

Examples

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

Related Topics

[\\$scwb deleteLayoutClbk](#)

[overlay delete](#)

layout droasis

Opens a disk-resident OASIS file in read-only mode.

Usage

layout droasis *filename*

Arguments

- *filename*

A required argument specifying the name of the OASIS file to load.

Return Values

The layout handle.

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.

You can also load a disk-resident OASIS file from the batch command line by specifying the layout viewer tool invocation with the *-v* option.

Examples

```
layout droasis hier.oas
```

layout filemerge

Performs a disk-based merge of multiple GDS files, multiple OASIS files, or a combination of GDS and OASIS files without loading them into memory.

Usage

```
layout filemerge {-in {layoutfile | directory | textfile |  
    {-name layoutfile  
        [-layerbump layer_bump  
        [-map_layer {{from_layer [.dt1] to_layer [.dt2]} ...}  
        {{[-exclude_layer {l1 [.dt] ... ln [.dtn]}]} | [-include_layer {l1 [.dt] ... ln [.dtn]}]}  
        [-prefix prefix  
        [-suffix suffix  
        [-exclude_top  
        [-excludeprefixsuffix {cell ...}  
        [-placecell [-refcell refcell] [-incell incell] [-x x] [-y y] [-mirror mirror]  
            [-angle angle] [-mag mag]}...  
        [-scale scale_value]}  
    }}...  
    {{-out output_file} | {-oasisout output_file} | {-gdsout output_file} |  
    {-indexout output_index_file} }  
    [-mode {append | overwrite | rename | forcerename | reportconflictsonly}]  
    [-cblockmode {0 | 1 | auto}]  
    [-compressionlevel {1 | 9}]  
    [-smartdiff {{[-ignoregeometries] [-ignoreproperties] [-ignorereferences] [-ignoretexts]  
        [-convertpathtopoly] [-compare_layers {l1 [.dt] l2 [.dt] ... ln [.dtn]}]}]}  
    [-noemptycells {0 | 1 | 2 | 3}]  
    [-createcache {0 | 1}]  
    {{[-exclude_layer {l1 [.dt] ... ln [.dtn]}]} | [-include_layer {l1 [.dt] ... ln [.dtn]}]}  
    [-createtop cellname]  
    [-topcell topcellname]  
    [-map_cell {{cellname mapped_ cellname} | filename}]  
    [-map_layer {{from_layer [.dt1] to_layer [.dt2]} ...}]  
    [-map_text {{cellname layer [.dt] from_text to_text} | filename} ]  
    [-peek [-cells] [-layers] [-undefcells]]  
    [-preserve filterfile]  
    [-strictmode {0 | 1}]  
    [-summary summary_filename]  
    [-tmp tmp_dir]  
    [-integerScaling]  
    [-precision output_precision]  
    [-reportconflicts]  
    [-stringSeverity {ignore | warning | fix | error}]  
    [-exclude_cells cells [-deleteChildCells]]
```

[-exclude_properties]
[-threads *num_threads*]

Arguments

- **-in** {*layoutfile* | *directory* | *textfile* | {-name *layoutfile* [*file-specific-options*]}}

A required argument specifying the input layout files to use for the merge operation. You can specify the **-in** argument set multiple times in any execution of the layout filemerge command with any combination of the valid arguments. When you specify multiple input files in which the input precisions do not match, the lowest common multiple of the input precisions is calculated. Each input layout is then magnified as it is read to ensure the user unit values are retained.

layoutfile — Specifies the name of a GDS or OASIS file to use as input for the merge operation. If you specify the same input file multiple times, Calibre DESIGNrev issues a warning similar to the following:

Warning: Input layout number <n>, filename '<name>' was already specified.

You can set the FILEMERGE_NO_WARN_DUPLICATE_LAYOUTS environment variable to prevent checking for duplicate input files.

directory — Specifies the name of a directory that contains one or more input files to be merged. All valid layout files in the specified directory are included in the merged output file; invalid layout files are ignored. Layouts in the specified directory are processed alphabetically. If multiple directories are specified as input, the directories are processed in the order specified and the layouts within their respective directories are processed alphabetically.

Note



With Calibre 2018.4, the **-in directory** argument set replaces the -indir argument. However, the use of -indir is still supported.

textfile — Specifies the name of an ASCII file that contains the names of the input layouts to be merged. The ASCII file can contain layouts and directories that contain layouts. The following rules apply to the use of an ASCII file:

- Empty lines and lines beginning with a pound sign (#) are ignored.
- Each layout or directory name must appear on a separate line in the file.
- A path to another text file is not allowed.
- File-specific arguments, such as -exclude_layer or -suffix, can be specified for a layout with the following conditions:
 - The layout name must appear first in the line before any file-specific arguments.
 - File-specific arguments cannot be specified for directories.

- The **-name** argument should not be used.

See “[Example 22 - Specify an ASCII File as Input](#)” on page 208.

-name layoutfile [*file-specific-options*] — Specifies the name of a GDS or OASIS file containing the layers or cells to filter or manipulate when merging with another file. Use this argument when you want to specify file-specific arguments, such as **-layerbump** or **-map_layer**.

The **-name layoutfile** argument set and associated values should be specified using the Tcl list command and be enclosed in brackets as shown in the following example:

```
layout filemerge -in layout1.gds -in [list -name layout2.gds] \  
-out layout3.gds
```

Note



With Calibre 2018.4, the **-in {-name layoutfile}** argument set replaces the **-infile** argument. However, the use of **-infile** is still supported.

The *file-specific-options* to the **-name layoutfile** argument set are:

-layerbump layer_bump — Specifies a value by which the layer number of each layer in **layoutfile** is incremented in the output file. This argument cannot be specified with **-map_layer** within the same **-name** argument set.

-map_layer { { from_layer[.dt1] to_layer[.dt2] } ... } — Specifies that objects on the *from_layer* of **layoutfile** are written to the *to_layer* in the output file. You can specify layers as either a layer number or a layer number and datatype value separated by a period (.). The **-map_layer** argument and associated *from_layer* and *to_layer* values should be specified using the Tcl list command and be enclosed in brackets as shown in the following example:

```
-map_layer [list 0 1 100.5 200.5]
```

You cannot specify **-map_layer** and **-layer_bump** within the same **-name** argument set. When you specify **-map_layer** with **-include_layer** or **-exclude_layer**, the layers specified by **-include_layer** or **-exclude_layer** are filtered before applying the **-map_layer** argument to any unfiltered layers.

This argument applies only to the layout specified by **-name layoutfile**. To map layers globally, use the **-map_layer** argument available with the **-in layoutfile**, **-in directory**, or **-in textfile** argument sets. Refer to “[Using -map_layer, -include_layer, and -exclude_layer](#)” on page 201 for more information.

-exclude_layer { ll[.dt] ... ln[.dtn] } — Specifies a list of layer numbers to exclude from the merged output file. You can specify layers as either a layer number or a layer number and datatype value separated by a period (.). The **-exclude_layer** argument can not be specified with **-include_layer**. The **-exclude_layer** values should be specified using the Tcl list command and enclosed in brackets as shown in the following example:

```
layout filemerge -in layout1.gds -in [list -name layout2.gds] \  
-exclude_layer [list 1 2 3]] -out layout3.gds
```


Instead of specifying layer numbers with the `-exclude_layer` argument, you can specify the name of an ASCII file that contains the layer numbers to be excluded from the merged output file. In the following example, the `oas_layers.txt` file contains a list of layers to be excluded from `layout1.oas` and the `gds_layers.txt` file includes a list of layers to be excluded from `layout2.gds`:

```
layout filemerge \  
  -in [list -name layout1.oas -exclude_layer oas_layers.txt] \  
  -in [list -name layout2.gds -exclude_layer gds_layers.txt] \  
  -out merged.oas
```

The layer numbers in the ASCII file can be on the same line or span multiple lines, as long as each number is separated by a space, tab, or newline character.

This argument applies only to an input file specified by **-name layoutfile**. To exclude layers globally, use the `-exclude_layer` argument available with the **-in layoutfile**, **-in directory**, or **-in textfile** argument sets. Refer to “[Using -map_layer, -include_layer, and -exclude_layer](#)” on page 201 for more information.

-include_layer { ll[.dt] ... ln[.dtn] } — Specifies a list of layer numbers to include in the merged output file. You can specify layers as either a layer number or a layer number and datatype value separated by a period (.). The `-include_layer` argument can not be specified with `-exclude_layer`. The `-include_layer` values should be specified using the Tcl list command and enclosed in brackets as shown in the following example:

```
layout filemerge -in layout4.gds -in [list -name layout5.gds \  
  -include_layer [list 1 2 3]] -out layout6.gds
```

Instead of specifying layer numbers with the `-include_layer` argument, you can specify the name of an ASCII file that contains the layer numbers to be included in the merged output file. In the following example, the `oas_layers.txt` file contains a list of layers to be included in `layout1.oas` and the `gds_layers.txt` file includes a list of layers to be included in `layout2.gds`:

```
layout filemerge \  
  -in [list -name layout1.oas -include_layer oas_layers.txt] \  
  -in [list -name layout2.gds -include_layer gds_layers.txt] \  
  -out merged.oas
```

The layer numbers in the ASCII file can be on the same line or span multiple lines, as long as each number is separated by a space, tab, or newline character.

This argument applies only to an input file specified by **-name layoutfile**. To include layers globally, use the `-include_layer` argument available with the **-in layoutfile** or the **-in directory** argument sets. Refer to “[Using -map_layer, -include_layer, and -exclude_layer](#)” on page 201 for more information.

-prefix prefix — Prepends the specified *prefix* to each cellname (including top cells) in order to create unique cellnames, thereby preventing cells with the same name from being merged. This argument is supported with all merge modes except overwrite mode (`-mode overwrite`). This argument generates an error when used with `-mode overwrite`, `-map_cell`, or the `-preserve` argument.

- suffix *suffix* — Appends the specified *suffix* to each cellname (including top cells) in order to create unique cellnames, thereby preventing cells with the same name from being merged. This argument is supported with all merge modes except overwrite mode (-mode overwrite). This argument generates an error when used with -mode overwrite, -map_cell, or the -preserve argument.
- exclude_top — Excludes the topcells in each input file from having a prefix or suffix appended. This argument can only be specified when using the append merge mode (-mode append) and must be specified for all or none of the input files.
- excludeprefixsuffix {*cell ...*} — Specifies the cell(s) in each input file that should be excluded from prepending a prefix or appending a suffix. This argument can only be used with the -prefix or -suffix arguments. The value specified for *cell* is a cellname, filename, or a Tcl list of cellnames. The following conditions apply:
 - When specifying a cellname, the cellname can be specified explicitly or using a wildcard (*, ?, or [] characters are supported).
 - When specifying a filename, the file must contain cellnames, one on each line, specified explicitly or using a wildcard.
 - When specifying a Tcl list, each cellname in the list must be specified explicitly or using a wildcard.
- placecell [-refcell *refcell*] [-incell *incell*] [-x *x*] [-y *y*] [-mirror *mirror*] [-angle *angle*] [-mag *mag*] ... — Specifies the location for placing a cell from the input file specified by **-name layoutfile** into the newly created top cell in the output file. You can specify the -placecell argument multiple times.
 - -refcell *refcell* — Specifies which cell from the input file is to be referenced. If this argument is not specified, then the topcell from the input layout is used. If there is more than one topcell, then one is selected and the transcript lists the topcell that was chosen in addition to the other available topcells.
 - -incell *incell* — Specifies the name of a cell in which to create a new placement of a reference from an input file. The following restrictions apply when using -incells:
 - The specified *incell* cannot exist in a layout specified with the -name argument, but must exist in at least one of the other specified input layouts. Different layouts can specify different *incell* values to -incell.
 - The -incell argument cannot be used with -mode forcerefname (or -forcerefname).
 - The -createtop argument is optional if -incell is specified at least once. If both -createtop and -incell are specified, the top cell for any input layout in which -incell is not specified is placed at the top level in the input layout. If -createtop is not specified, then the top cell for any input layout that does not use -incell is placed at the top level of the output layout.

- When `-refcell` is not specified with `-incell`, the default is to use a topcell from the layout specified by the `-name` argument.
- `-x x` — Specifies an *x* coordinate value at which the origin of the referenced cell is placed in the merged output file. The default coordinate value is 0. A coordinate can be specified as a dbu (default) or micron value.

A dbu value can be specified as an integer that is followed by a “d” to indicate dbu, or as a floating point number as long as the number contains no fractional component (for example, 1024.0). Specifying a fractional component for a dbu value will generate an error. For example:

```
Error converting the -x argument value, 10000.1, to an integer.
```

A micron value can be specified as a floating point number or an integer, and must be followed by a “u” to indicate micron.

- `-y y` — Specifies a *y* coordinate value at which the origin of the referenced cell is placed in the merged output file. The default coordinate value is 0. A coordinate can be specified as a dbu (default) or micron value.

A dbu value can be specified as an integer followed by a “d” to indicate dbu, or as a floating point number as long as the number contains no fractional component (for example, 1024.0). Specifying a fractional component for a dbu value will generate an error. For example:

```
Error converting the -x argument value, 10000.1, to an integer.
```

A micron value can be specified as a floating point number or an integer, and must be followed by a “u” to indicate micron.

- `-mirror mirror` — Specifies whether the referenced cell is mirrored in the merged output file. Valid values for *mirror* are:
 - `true` — The referenced cell is mirrored along the x-axis in the merged output file. To mirror along the y-axis, specify both “`-mirror true`” and “`-angle 180`”.
 - `false` — The referenced cell is not mirrored in the merged output file. This is the default.
- `-angle angle` — Specifies an angle at which to place the referenced cell in the merged output file. The default angle is 0.
- `-mag mag` — Specifies a value for magnifying the referenced cell in the merged output file. The default magnification is 1.0.

Specify the `-placecell` argument using the Tcl list command and enclose associated values in brackets as shown in the following example:

```
layout filemerge -in spare_top_fill.oas \  
  -in [list -name spare_top_wIP.oas \  
    -placecell [list -refcell DM1_5c]] -createtop new_top \  
  -out ./output/out.oas
```

The following requirements and general information regard the use of the `-placecell` argument:

- The `-topcell` argument cannot be used with `-placecell`.
- The `-createtop` argument is required when specifying `-placecell`. Cells are placed in the merged results under the topcell specified by `-createtop`.
- You can specify the `-placecell` argument multiple times within a list of arguments that are passed to the **-name *layoutfile*** argument set as shown in the following example:

```
layout filemerge -in spare_top_fill.oas \  
  -in [list -name spare_top_wIP.oas \  
    -placecell [list -refcell topcell_1] \  
    -placecell [list -refcell topcell_2]] -createtop new_top \  
  -out ./output/out.oas
```

This is useful when a layout contains more than one top cell and you want to preserve each of the top cells.

- You can specify to place the *same* cell multiple times from any input layout. For example:

```
layout filemerge -in [list -name layout.oas \  
  -placecell [list -refcell CELL88] \  
  -placecell [list -refcell CELL88 -x -10 -y -10]] \  
  -out CELL88.oas -createtop new_top
```

- When using `-placecell` with `-mode rename` or `-mode forcereplace`, each cell is uniquely named so that it is placed at the specified location in the merged output. However, when using `-placecell` with `-mode append` or `-mode overwrite`, the results of each appended or overwritten cell are placed at the specified location and, as a result, the appended or overwritten cell in the merged output may not match the original cell in the input layout.
- When `-placecell` is not specified for all input layouts (in the same layout filemerge operation), then for each input layout for which `-placecell` is specified, only the specified cells and associated hierarchy are placed into the new topcell. The topcells and children in the layouts for which `-placecell` is not specified, are written as children of the new topcell specified by `-createtop`. In the following example, the topcells from *layout0.oas*, *layout1.oas*, and *layout2.oas* are placed in the resulting output under the new topcell specified by the `-createtop` argument (NEWTOP).

```
layout filemerge -in layout0.oas -in layout1.oas \  
  -in [list -name layout2.oas -placecell { -x 50u -y 50u }] \  
  -out ./output/merged.oas -mode rename -createtop NEWTOP
```

- When an input layout contains more than one topcell and the `-placecell` argument is not specified, the layout filemerge operation includes each topcell at the top level of the resulting output and generates a note similar to the following:

Note: Layout "layout.oas" contains more than one top cell, all were implicitly placed: TOPCELL_A, TOPCELL_B.

`-scale scale_value` — Specifies a positive real number by which to scale the input layout. To scale up a layout, specify a value greater than 1.0. To scale down a layout, specify a value greater than 0 and less than 1.0 (for example, 0.5).

- `{-out output_file} | {-oasisout output_file} | {-gdsout output_file} | {-indexout output_index_file}`

A required argument specifying a filename in which to output the results of the merge operation. The name specified for *output_file* cannot be the same as any of the input filenames.

`-out output_file` — Specifies the name of the file in which to output the results. When this argument is used, the output format of the merged files is determined by the input file format, as follows:

- Merged GDS input files output GDS format.
- Merged OASIS input files output OASIS format.
- Mixed input files (GDS and OASIS) output OASIS format.

`-oasisout output_file` — Specifies the name of an OASIS file in which to output the results. This argument is useful when all of the input layouts are GDS format and you want the merged results in OASIS format.

`-gdsout output_file` — Specifies the name of a GDS file in which to output the results. Any input layouts that are in OASIS format are converted to GDS prior to merging.

`-indexout output_index_file` — Specifies the name of a file in which to output cellname information. The index file generated by this argument is not a valid OASIS or GDS layout, but can be used as input layout to a subsequent layout filemerge command. Some layout peek arguments, such as `-format`, `-cells`, and `-topcell`, can be used to return information from the index file.

The `-out`, `-oasisout`, and `-gdsout` arguments are mutually exclusive and each argument cannot be specified with the `-mode reportconflictsonly` argument. Specifying `-out`, `-oasisout`, or `-gdsout` with `-mode reportconflictsonly` generates an error.

The merged results can be compressed to a gzip file by specifying a `.gz` (or `.GZ`) suffix for the output filename. The `-cblockmode` and `-compressionlevel` arguments have no effect when compressing an OASIS file to a zipped output file. Specifying the output as zipped

OASIS with the `-strictmode` argument generates the following warning and disables strict mode:

```
Warning: Strict mode is not supported when writing a gzipped output
layout, strict mode disabled.
```

- `-mode {append | overwrite | rename | forcereaname | reportconflictsonly}`

An optional argument specifying how duplicate cells (cells in the input files having the same name) are handled. When a merge mode is not specified, the default is to append cells that have the same name. See “[Managing Cellname Conflicts](#)” on page 202 for help handling cell merge conflicts.

`append` — Concatenates the contents of the duplicate cells together, including all references and objects. If the `-smartdiff` argument is specified, action is taken only if the cell contents are different.

`overwrite` — Writes the contents of duplicate cells from the second file over the contents of the first, deleting the contents of the cell from the first file. This argument prints a warning message when specified with the `-smartdiff` argument.

`rename` — Renames the duplicate cell, to make it unique. If the `-smartdiff` argument is specified, action is taken only if the cell contents are different.

`forcereaname` — Renames the original and new cellnames to be unique, by adding a `_WBx` extension, where `x` is an integer. When used with the `-prefix` or `-suffix` argument, all cellnames are renamed using the specified prefix, suffix, or both. This argument prints a warning message when specified with the `-smartdiff` argument.

`reportconflictsonly` — Reports any cells in the input files that have the same name. When it is not obvious which merge mode (`append`, `overwrite`, `rename`, or `forcereaname`) is most appropriate for merging files, you can use this option to report cellname conflicts, thereby allowing you to determine which mode to use when merging the files.

Specifying this option with `-reportconflicts` generates a warning and the option specified last overrides the option specified first.


Unlike the `-reportconflicts` argument, `-reportconflictsonly` does not support writing to an output file and does not provide information on how cell naming conflicts are resolved. Attempting to use this argument with `-out` issues a warning similar to the following:

```
Warning: The -out switch and argument '<output_file>.oas' will be
ignored
due to the use of the -reportconflictsonly mode.
```

When `-smartdiff` is specified with `-reportconflictsonly`, cells having the same name and different content (geometries, placements, properties, text) are reported. Cells having the same name and content are not reported.

The `reportconflictsonly` option disables the `-strictmode` argument.

Note

 You can specify each of these options without the -mode keyword. For example, -append, -overwrite, -rename, -forcerename, and -reportconflictsonly are all valid arguments.

- -cblockmode {0 | 1 | auto}

An optional argument that writes OASIS layout files using CBLOCK compression. CBLOCKS are only written for cell records and strict mode tables that are larger than 128 bytes.

0 — CBLOCKS are not written to the output file.

1 — CBLOCKS are written to the output file. This is the default.

auto — If any of the input files contain CBLOCKS, then the CBLOCKS are written to the output file.

When the output format is a gzipped OASIS file (.GZ or .gz), specifying this argument generates a warning and does not write CBLOCKS to the file.

- -compressionlevel { 1 | 9 }

An optional argument that specifies the level of CBLOCK compression used when writing OASIS layout files. This argument only has an effect if -cblockmode is also specified.

1 — Uses the highest CBLOCK compression speed.

9 — Uses the highest CBLOCK compression rate.

The value specified with this argument is used instead of the value set in the Preferences dialog box. When this argument is not specified, the filemerge command uses the value set in the Preferences dialog box.

When the output format is a gzipped OASIS file (.GZ or .gz), this argument has no effect.

- -smartdiff [-ignoregeometries] [-ignoreproperties] [-ignorereferences] [-ignoretexts] [-convertpathtopoly] [-compare_layers {l1[.dt] l2[.dt] ... ln[.dtn]}]

An optional argument that specifies to perform additional processing when cellname conflicts occur. In the case of a conflict, the command compares the cell topologies and only alters cells when the cell contents are different. This argument provides a cleaner hierarchy for the merged output file and minimizes the file size. You can control the behavior of the -smartdiff operation by specifying one or more of the following arguments:

-ignoregeometries — Specifies that geometries are not considered for comparison.

-ignoreproperties — Specifies that properties are not considered for comparison.

-ignorereferences — Specifies that cell references are not considered for comparison.

-ignoretexts — Specifies that text objects are not considered for comparison. The text, layer, datatype, and x/y values are considered for comparison. Any attributes are ignored.

- convertpathtopoly — Specifies that paths are converted to polygons for comparison. The default is to not convert paths to polygons.
- compare_layers {I1[.dt] I2[.dt] ... In[.dtn]} — Specifies that only the objects on the specified layers are considered for comparison when checking for differences between cells. You can specify layers as either a layer number or a layer number and datatype value separated by a period (.).

The -smartdiff arguments must be specified as a Tcl list enclosed in brackets. For example:

```
layout filemerge -in layout1.gds -in layout2.gds \  
-smartdiff [list -ignoretexts -convertpathtopoly] \  
-out layout3.gds
```

Using -smartdiff with the -mode forcereaname (or -forcereaname), -mode overwrite (or -overwrite), -prefix, or -suffix argument eliminates the ability to detect cell conflicts and impacts performance. Use of these arguments with -smartdiff issues a warning. For example:

- o Using -smartdiff with -mode forcereaname:

```
Warning: Using the "-smartdiff" switch in combination with the  
"-mode forcereaname" switch  
may impact the performance of filemerge without changing the  
output.
```


- o Using -smartdiff with the -mode overwrite:

```
Warning: Using the "-smartdiff" switch in combination with the  
"-mode overwrite"  
switch may impact the performance of filemerge without changing  
the output.
```

- o Using -smartdiff with the -suffix argument:

```
Warning: Using the "-smartdiff" switch in combination with the  
"-in {-suffix...}" switch(es)  
may impact the performance of filemerge without changing the  
output.
```

Note

 When using -smartdiff with a file-specific -include_layer or -exclude_layer argument, Calibre DESIGNrev performs the smartdiff comparison before filtering any layers. To perform file-specific layer filtering prior to performing the smartdiff comparison, you must run layout filemerge twice, once to filter layers and the second time to perform the smartdiff comparison. The global -include_layer and -exclude_layer arguments do not affect the smartdiff comparison, since the same layers are filtered from all cells.

- -noemptycells {0 | 1 | 2 | 3}

An optional argument used to control the writing of cell definitions for empty cells and references to empty cells.

- 0 — Writes empty cell definitions to the output file. This is the default.
- 1 — Does not write empty cells to the output. References to empty cells are written to the output.
- 2 — Does not write empty cells or their immediate references to the output. Cells that contain only references to empty cells are written to the output.
- 3 — Does not write empty cells or their references to the output. This value works recursively, removing all cells that only contain references to empty cells.

- **-createcache {0 | 1}**

An optional argument specifying whether to create a PCR (cache) file for the merged output file.

0 — No cache file is created. This is the default.

1 — A cache file is created for the generated merged output file.

The PCR file is used for performing incremental loading of a layout and can speed up initial loading of a layout. Refer to “[The Peek Cache Repository File](#)” and “[Incrementally Loading Layouts](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for more information.

The -createcache argument disables the -strictmode argument. Refer to “[Creating a PCR File for an OASIS Strict Mode Layout](#)” on page 202 for more information.

- **-exclude_layer {ll[.dt] ... ln[.dtn]} | -include_layer {ll[.dt] ... ln[.dtn]}**

An optional argument specifying a list of layer numbers that contain the objects to exclude or include from the merged output file. You can specify layers as either a layer number or a layer number and datatype value separated by a “.”. The -exclude_layer or -include_layer argument and associated values should be specified using the Tcl list command and be enclosed in brackets. For example:

```
layout filemerge -in layout1.gds -in layout2.gds \  
-exclude_layer [list 1 2 3] -out layout3.gds
```

Instead of specifying layer numbers, you can specify the name of an ASCII file containing the layer numbers to be excluded or included from the merged output file. For example:

```
layout filemerge -in layout0.gds -in layout1.gds -out merged.gds \  
-exclude_layer exclude_layers.txt
```

The layer numbers in the ASCII file can be on the same line or span multiple lines, as long as each number is separated by a space, tab, or newline character.

The -exclude_layer and -include_layer arguments apply globally to any input files specified by the **-in layoutfile**, **-in directory**, and **-in textfile** argument sets. To exclude or include layers from a specific input file, use the -exclude_layer or -include_layer argument with the **-in -name layoutfile** argument set. Refer to “[Using -map_layer, -include_layer, and -exclude_layer](#)” on page 201 for more information on using this argument.

- **-createtop *cellname***

An optional argument that specifies the name of a topcell in which to place the topcells from the input files. During the merge operation, the topcell in each input file becomes a child of the new topcell. When this argument is not specified, the input layouts are merged in import-style and no topcell is created.

When this argument is specified with **-topcell**, the value specified for *cellname* becomes the new top cell and it contains the cell specified by *topcellname*.

- **-topcell *topcellname***

An optional argument that specifies the name of a top cell to output to the resulting merged layout. Only one top cell and the references in the hierarchy of the top cell are output to the merged results. This argument can be useful in removing unreferenced cells from an IP library file.

- **-map_cell {{*cellname mapcellname*} | *filename*}**

An optional argument that renames (or maps) a cellname from the input layout file(s) to a cellname in the output file. You can specify the **-map_cell** argument multiple times in the same layout filemerge command, using either form of the mapping options. If you specify multiple mappings of the same cellname, the first mapping is always used. The cells are renamed before merging.

cellname mapcellname — Specifies a cellname in the input file (*cellname*) and the cellname to map it to in the output file (*mapcellname*).

filename — Specifies the name of an ASCII file containing the cellname mappings. The cellname for each “from” cell and “to” cell is specified on a separate line in the file. Blank lines and lines beginning with the pound sign (#) are ignored.

Both mapping options support the use of glob-style wildcards (for example, *, ?, and [...]) for the cellname in the input file. In this example, the wildcards match cells named “CELL3abc” and “CELL5xyz” in an input layout file.

```
CELL[35]??? NEW_MAPPED_CELL
```

Because wildcards can match zero or more cell occurrences, you should ensure they are only used for cells that are identical but have different names, as only one cell will be written to the output file.

Note



The intended purpose of the **-map_cell** argument is to make it easy to remove many similarly named child cells, by giving the child cells the same name, and then deleting those cells. However, renaming *both* parent and child cells to the same name can create circular references.

- **-map_layer {{*from_layer* [*.dt1*] *to_layer* [*.dt2*]} ...}**

An optional argument specifying that objects on the *from_layer* of the input file are written to the *to_layer* in the resulting output file. Layers can be specified as either a layer number

or a layer number and datatype value separated by a period (.). You should use the Tcl list format to specify multiple *from_layer* and *to_layer* values. For example:

```
-map_layer [list 0 1 100.5 200.5]
```

You cannot specify `-map_layer` and `-layer_bump` for files specified by the `-in` argument. When you specify `-map_layer` with `-include_layer` or `-exclude_layer`, the layers specified by `-include_layer` or `-exclude_layer` are filtered before applying the `-map_layer` argument to any unfiltered layers.

This argument applies globally to any input files specified by the **-in layoutfile**, **-in directory**, and **-in textfile** argument sets. To perform file-specific layer mapping, use the `-map_layer` argument available with the **-in -name layoutfile** argument set. Refer to “[Using -map_layer, -include_layer, and -exclude_layer](#)” on page 201 for more information on using this argument.

- `-map_text { {cellname layer[.dt] from_text to_text} | filename }`

An optional argument that renames (or maps) text in the output file.

cellname layer[.dt] from_text to_text — Specifies to rename all instances of *from_text* in the specified cell and layer to *to_text*.

filename — Specifies the name of an ASCII file containing the text mappings. The *cellname*, *layer*, “from” text, and “to” text for each text mapping is specified on a separate line in the file. Blank lines and lines beginning with the pound sign (#) are ignored.

Mapping statements use the following syntax:

```
cellname layer[.datatype] from_text to_text
```

Both mapping options support the use of glob-style wildcards (for example, *, ?, and [...]) for the *cellname* in the input file.

- `-peek [-cells] [-layers] [-undefcells]`

An optional argument specifying the information to return from the layout filemerge operation. This argument returns values in the same format as the [layout peek](#) command. Values returned by this argument can be assigned to a Tcl variable for further processing.

`-cells` — Returns the names of all cells in the merged results.

`-layer` — Returns the names of all layers in the merged results.

`-undefcell` — Returns the names of all cells that are referenced but not defined in the merged results.

To return results from more than one `-peek` argument, the arguments must be grouped in a Tcl list format. For example:

```
-peek [list -cells -layers]
```

- **-preserve *filterfile***

An optional argument specifying the name of a file that contains the filenames and cellnames to preserve during the merge operation. This argument makes controlled merges possible by preserving the layout file and cell pairs in the merged output file. In other words, the cellnames in a specific layout file will always overwrite cells with the same name from any input layout files during the merge operation. This can be useful for assembling library files.

The syntax of the *filterfile* is as follows:

```
filename [cellname | -topcell] [-nohier]
```


- *filename* — Specifies the full or relative pathname of the layout.
- *cellname* | -topcell — Specifies the name of the cell in the layout that you want to preserve. You can specify “-topcell” instead of a *cellname*, which preserves the topcell in the specified layout. A *cellname* can not be specified with -topcell.
- -nohier — Preserves only the specified cell. When this argument is not specified, all cells are preserved down the hierarchy of a cell specified in the *filterfile*.

Each *filename* and *cellname* (or -topcell) that you want to preserve must be specified on a separate line in the *filterfile*. For example:

```
filename_x cellname_a  
filename_x cellname_b -nohier  
filename_z -topcell
```

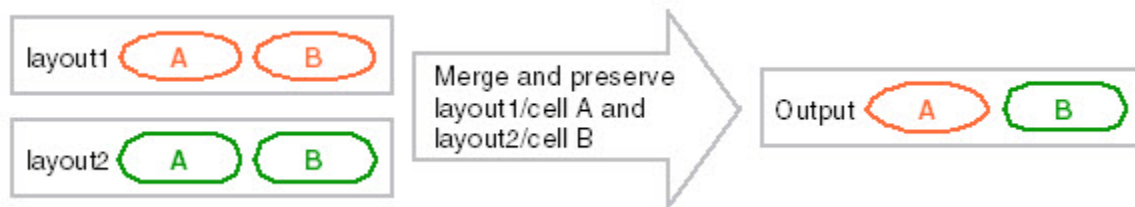
If a *cellname* is specified more than once with a different *filename*, then the last *filename* and *cellname* definition in the *filterfile* takes precedence.

Note

 If -preserve is not specified and a cellname conflict is discovered during the merge operation, then the last input file on the command line appends, overwrites, or renames cells in the other input file(s). When a cellname conflict occurs during overwrite mode, the last input file specified on the command line overwrites cells from other input files.

For example, consider the case where you want to use cell A from layout 1 and cell B from layout 2, but both layout 1 and layout 2 have a cell A and cell B. Without the -preserve argument, you can only have A and B in the output from the same layout. Using the -preserve argument you can preserve the cells in the final result by specifying cell A, layout 1 and cell B, layout 2 in the *filterfile*, as follows:

```
layout1 A  
layout2 B
```

Figure 5-2. Merge and Preserve

- `-strictmode {0 | 1}`

An optional argument that specifies whether or not to create strict mode output for the resulting OASIS layout database.

0 — Creates non-strict mode output.

1 — Creates strict mode output. This is the default.

When creating a gzipped OASIS file and `-strictmode 1` (the default) is specified, a warning is generated and strictmode is disabled.

The `-strictmode` argument cannot be enabled with the `-createcache` argument. Refer to “[Creating a PCR File for an OASIS Strict Mode Layout](#)” on page 202 for more information.

For more information on OASIS strict mode, refer to the section “[Name Records and Table Offsets](#)” in the *Open Artwork System Interchange Standard (OASIS) Guide*.

- `-summary summary_filename`

An optional argument that specifies to output summary information from the merge operation. The summary information is written to the *summary_filename* when it is specified; otherwise, the summary information is written to standard output. The summary information includes the command line followed by a list of cell information, including the input layout, input cell, occurrence, output cell, and (if applicable) the reason why it was omitted from the output layout. Both the command line and cell information are enclosed in xml-format markers and the contents of the body are in CSV format. For example:

```
<command_line>
layout filemerge -in layout1.oas -in layout2.oas
-out merged_results.oas -summary summary
</command_line>
<cell_summary>
Input Layout,Input Cell,Occurrence,Output Cell,Reason
layout1.oas,cell1,1,cell1,
layout2.oas,cell2,1,cell2,
...
</cell_summary>
```

- `-tmp tmp_dir`

An optional argument used to specify a temporary directory for expanding input files that are in gzip format. This argument is useful for performance reasons.

If you do not specify this argument, the merge operation uses the value of the MGC_CWB_TMP_DIR environment variable, or the MGC_TMPDIR environment variable if MGC_CWB_TMP_DIR is not set.

- **-integerScaling**

An optional argument used to set the target output precision to the lowest common multiple of all input layout files. A positive integer value is used to scale the precisions instead of a fractional number, which can potentially cause problems with some geometry representations.

The default behavior is to scale the input layout files having lower precision values to match that of the layout file with the highest precision value. For example, if you merge two layout files whose precisions are 4000 and 10000, the layout with the precision of 4000 is scaled by 2.5 so that it also has a precision of 10000.

Using this same example, if -integerScaling is specified, the layout file with a precision of 4000 is scaled by 5 and the layout file with a precision of 10000 is scaled by 2, which avoids scaling by fractional numbers.

This argument cannot be specified with -precision.

- **-precision *value***

An optional argument used to set the target output precision of the merged results. The specified *value* must be greater than 0.

This argument cannot be specified with -integerScaling.

- **-reportconflicts**

An optional argument that reports merge conflicts. Information generated indicates cells causing name conflicts as well as actions taken for each merge conflict. For example:

- Using -reportconflicts with -mode append and -smartdiff:

Note: cell spare_top already exists in spare_top_fill.oas and content differs,
appended with spare_top from spare_top_wIP.oas.

- Using -reportconflicts with the -mode rename and -smartdiff:

Note: cell spare_top already exists in spare_top_fill.oas and content differs,
renamed with spare_top_WB1 from spare_top_wIP.oas.

- Using -reportconflicts with -mode overwrite:

Note: cell spare_top already exists in spare_top_fill.oas and content differs,
overwritten with spare_top from spare_top_wIP.oas.

- Using -reportconflicts with -mode forcerename:

Note: cell spare_top already exists in spare_top_fill.oas and content differs,
force renamed with spare_top_WB2 from spare_top_WIP.oas.

- -stringSeverity {ignore | warning | fix | error}

An optional argument that specifies the type of message, if any, that is generated when illegal a-string values are found while reading an OASIS file.

ignore — Does not generate any messages when illegal a-string values are found while reading an OASIS file, and writes the output layout with illegal characters intact.

warning — Generates a warning message for each illegal a-string value found while reading an OASIS file, and writes the output layout with illegal characters intact.

fix — Replaces any illegal a-string character with a space () character and generates a warning message.

error — Generates an error message for each illegal a-string value found while reading an OASIS file, and does not output an OASIS file. This is the default.

- -exclude_cells *cells* [-deleteChildCells]

An optional argument that specifies the name of one or more *cells* to suppress from being output by the filemerge operation. This argument can be specified one or more times per invocation of the layout filemerge command.

Any references to the excluded cells are output to the merged results but the cells are left undefined. When loading the merged results, Calibre DESIGNrev replaces any undefined cells with empty cells. To suppress references to excluded cells, use the -noemptycells argument with the 2 or 3 value.

-deleteChildCells — Deletes any child cells that are referenced in the hierarchy of the specified cell(s). Child cells referenced outside of the hierarchy of the specified cell are not deleted.

The following rules apply to cell names:

- Cell names can be specified explicitly or by using wildcard characters.
- Glob-style wildcards (*, ?, \, and [...]) are supported in cell names, where:
 - * — Matches any sequence of characters.
 - ? — Matches any single character.
 - \ — Escapes the special meaning of the wildcard character.
 - [...] — Matches the set of characters specified within the brackets. The dash (-) character can be used to denote a range.

- Multiple cells should be specified using the Tcl list command. The Tcl list command must be enclosed in brackets as shown in the following example:

```
layout filemerge -in layout1.gds -in layout2.gds \  
-out layout3.gds -exclude_cell [list cell_A cell_B cell_C]
```

- **-exclude_properties**

An optional argument specifying that geometry and placement properties should not be written to the merged layout file. Cell and file level properties are still written to the merged layout file.

- **-threads *num_threads***

Specifies the number of threads to use for running the merge operation. When this argument is not specified, the application uses the maximum number of CPUs that are available.


Return Values

Returns an error string when an error occurs. Otherwise, no values are returned when the files are successfully merged.

Description

The layout filemerge command performs a disk-based merge of multiple GDS input files, multiple OASIS input files, or a mixed input format (GDS and OASIS) without loading them into memory. Because the layout filemerge command does not load input files into memory, layout exceptions defined in the *readerprefs* file are ignored.

Note

 Layout filemerge is the recommended solution when merging files, over other commands such as layout merge or \$L import layout, because it is fast and has a low memory requirement.

Refer to the -out argument for information on the output format. The functionality provided by the layout filemerge command is also available in the Calibre DESIGNrev GUI by selecting **File > File Merge**. Refer to “[Layout Filemerge Dialog Box](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for more information.

The layout filemerge operation supports an import-style (one topcell is maintained) and merge-style (each file is given its own topcell) in the resulting output.

Specifying an input file with no additional arguments generates an output file as follows:

- The contents of duplicate cells are appended.
- CBLOCKS are written if any of the input OASIS files contain CBLOCKS.
- If there are cellname conflicts, cells are altered only when the cell contents are different.
- Empty cell definitions are written.

- The input layout(s) with lower precision values are scaled to match that of the layout file with the highest precision.

Files are processed in the order in which they are specified on the command line. For example, the files and directories specified in the following command:

```
layout filemerge -in layout0.oas -in dir1 \  
-in [list -name layout1.oas] -in dir2 -in layout2.oas \  
-out layout_merged.oas -report_conflicts
```

Are processed in the following order:

1. *layout0.oas*
2. Contents of *dir1*
3. *layout1.oas*
4. Contents of *dir2*
5. *layout2.oas*

The layout filemerge command supports gzipped input files and supports options for specifying where to expand input files that are in gzip format. Refer to the description of the `-tmp` argument for more information.

When merging a file that contains an undefined cell, the definition of the cell is taken from the first occurrence of the input file specified in the command that contains the cell definition. This behavior only applies when rename mode (`-rename`) is specified. For example, assume Cell A is referenced in *file1.gds* but is not defined, and it is defined in both *file2.gds* and *file3.gds*. If you specify the following command, the definition of cell A in the output file is taken from *file2.gds* since this is the first file specified in the command line that contains the cell definition:

```
calibredrv -a layout filemerge -in file1.gds -in file2.gds -in file3.gds \  
-rename -out out.gds
```

Using `-map_layer`, `-include_layer`, and `-exclude_layer`

You can apply the `-map_layer`, `-include_layer`, and `-exclude_layer` arguments globally to input files, at a file-specific level, or both globally and file-specific.

- To apply globally, you can specify any of these arguments with one or more of the **`-in layoutfile`**, **`-in directory`**, and **`-in textfile`** argument sets as shown in the following example:

```
layout filemerge -in layout0.oas -in layout1.oas \  
-in /home/layout_files -map_layer [list 111 222] \  
-include_layer [list 222]
```

- To apply at a file-specific level, you can specify any of these arguments with the **`-in -name layoutfile`** argument set. The `-map_layer`, `-include_layer`, and `-exclude_layer`

arguments are applied only to the associated input file specified by the **-in -name layoutfile** argument set. For example:

```
layout filemerge -in [list -name layout0.oas] \  
-maplayer [list 111 222] -include_layer [list 222]
```

- To apply both globally and at a file-specific level, you can specify any of these arguments with any combination of the **-in layoutfile**, **-in directory**, or **-in textfile** argument set. In this situation, the following rules apply:
 - The **-map_layer** values specified with **-in -name layoutfile** take precedence over the **-map_layer** values specified with the **-in layoutfile**, **-in directory**, or **-in textfile**.
 - The **-exclude_layer** or **-include_layer** values specified with **-in layoutfile** or **-in directory**, or **-in textfile** are not applied to a layout file for which a file-specific **-exclude_layer** or **-include_layer** argument is specified. For example, the following command excludes layer 1 from the results of merging *a.gds* to *out.gds* and excludes layer 0 from the results of merging *b.gds* to *out.gds*:

```
layout filemerge -in a.gds \  
-in [list -name b.gds -exclude_layer 0] -exclude_layer 1 \  
-out out.gds
```

Merging Databases with Different Precisions

You can merge files that have different precisions. The precision of the output file is the same as the highest precision among the input files. The x and y coordinates of records in the files at lower precision are magnified. A message appears in the transcript for each file that is scaled by the magnification factor.

To override the default behavior for determining the output layout precision, you can use the **-precision** or **-integerScaling** argument.

Creating a PCR File for an OASIS Strict Mode Layout

Specifying the **-createcache** argument with the **-strictmode** argument is not supported and generates an error. If you require a PCR file for an OASIS strict mode layout, you can create one by incrementally loading the strict mode layout using the **-incr** argument available with the **layout create** command. Refer to “[layout create \(GDS or OASIS file\)](#)” on page 161 for more information. You can also use the **-handle** argument available with the **layout peek** command to create a .pcr file for an OASIS strict mode layout. Refer to “[layout peek](#)” on page 221 for more information.

Managing Cellname Conflicts

By default, the **layout filemerge** command appends cells that have the same name. You can use either of the following methods to prevent cells with the same name from being merged:

- Specify the **-rename**, **-suffix**, or **-prefix** arguments to rename duplicate cells when a cellname conflict occurs.

- Specify the `-smartdiff` argument to only merge cells when their content differs.

You can also specify the `-reportconflicts` argument to output any merge conflicts.

Examples

Example 1 - Using Defaults

This example merges two layouts, *spare_top_fill.oas* and *spare_top_wIP.oas*, and outputs the results to *spare_top_merged.oas*. Default values for all arguments are used since no arguments are specified. Duplicate cells are concatenated, and CBLOCKS and empty cell definitions are written to the output. The input file with the lowest precision value is scaled to match the precision of the input file having the higher precision value. An error is generated if the databases have different precision values.

```
layout filemerge -in spare_top_fill.oas -in spare_top_wIP.oas \  
-out ./output/spare_top_merged.oas
```

Example 2 - Increment (or Bump) Layers

This example merges *spare_top_fill.oas* and *spare_top_wIP.oas* into the output file *spare_top_merged_bumped.oas*. All layers in *spare_top_wIP.oas* are incremented by 200. For example, layer 31 in *spare_top_wIP.oas* becomes layer 231 in *spare_top_merged_bumped.oas*.

```
layout filemerge -in spare_top_fill.oas -in spare_top_wIP.oas 200 \  
-out ./output/spare_top_merged_bumped.oas
```

Example 3 - Include Layers

This example merges *spare_top_fill.oas* and *spare_top_wIP.oas* into the output file *spare_top_merged_fill_layers.oas*. The `-in -name` argument set merges the layers 52.71, 53.71, 54.11, and 55.11 in *spare_top_fill.oas* with all layers in *spare_top_wIP.oas*.

```
layout filemerge -in [list -name spare_top_fill.oas \  
-include_layer [list 52.71 53.71 54.11 55.11]] \  
-in [list -name spare_top_wIP.oas] \  
-out ./output/spare_top_merged_fill_layers.oas
```

Example 4 - Filter Input Layers Specified in an ASCII File

This example merges the layout files named *layout0.oas* and *layout1.oas*. The *layers.txt* file specifies the layers to include in the merged output from *layout0.oas*. For example:

```
1.0 3.0 5.0 7.0 9.1 9.3
```

Each layout contains one cell named “TOPCELL”. The results are merged into “TOPCELL” in *out.oas*, and include the objects from all layers for *layout1.oas* and the objects from the layers specified in *include_layers.txt* for *layout0.oas*.

```
layout filemerge -in layout1.oas -in [list -name layout0.oas \  
-include_layer layers.txt] -out ./output/out.oas
```

Example 5 - Include All References Without Polygons or Texts

This example assumes there are only references (no objects) on layer 10 of the input GDS file named *lab1a.gds*. The following command outputs only the cellnames in *lab1a.gds* and the references on layer 10 to the file *out_layer10_refs.gds*.

```
layout filemerge -in lab1a.gds -include_layer 10 \  
-out ./output/out_layer10_refs.gds
```

Example 6 - Exclude Layers

This example merges the files *layout.gds* and *layout_block.gds*, excluding layers 1 and 2 in *layout_block.gds* from the results. The *-mode overwrite* argument causes duplicate cells from *layout_block.gds* to overwrite the same cell found in *layout.gds*. The results are output to *layout_exclude_layers.gds*.

```
layout filemerge -in layout.gds -in layout_block.gds \  
-exclude_layer [list 1 2] -out ./output/layout_exclude_layers.gds \  
-mode overwrite
```

Example 7 - Append a Specified Suffix

This example merges the layout files named *spare_top_fill.oas* and *spare_top_wIP.oas*. The merged results contain two top level cells named “spare_top_fill” and “spare_top_IP”. The string “_fill” is appended to each cell in *spare_top_fill.oas* and the string “_IP” is appended to each cell in *spare_top_wIP.oas*.

```
layout filemerge -in [list -name spare_top_fill.oas -suffix _fill] \  
-in [list -name spare_top_wIP.oas -suffix _IP] \  
-out merged_results.oas
```

Example 8 - Place Cells from Multiple Input Layout Databases

This example creates a new topcell named “new_top” and places the cells from three input layout databases (*layout1.gds*, *layout2.gds*, and *layout3.gds*) in *merged_layouts.gds* at the specified x and y coordinates. The x and y coordinates for *layout1.gds* are in dbus. The x coordinate value is specified as a floating point number that is acceptable since the number (10000.0) contains no fractional component. The x and y coordinates for *layout2.gds* and *layout3.gds* are specified in microns (indicated by the ‘u’ character), where some values are specified as floating point numbers with fractional components.

```
layout filemerge -createtop new_top \  
-in {-name layout1.gds -placecell {-x 10000.0 -y 20000d}} \  
-in {-name layout2.gds -placecell {-x 2.0u -y 1.5u}} \  
-in {-name layout3.gds -placecell {-x 3u -y 1.5u}} \  
-out merged_layouts.gds
```

Example 9 - Append or Concatenate Cells (to resolve cellname conflicts when merging)

This example combines the files *layout.gds* and *layout_block.gds* into the output file *layout_design_with_block.gds*. Any cells having the same name are concatenated.

```
layout filemerge -in layout.gds -in layout_block.gds \  
-out layout_design_with_block.gds -mode append
```

Example 10 - Overwrite or Swap Cells (to swap cell definitions)

In this example, any cells that have the same name are overwritten. Duplicate cells in *layout_block.gds* overwrite cells in *layout.gds*, deleting the contents of any duplicate cells from the first file.

```
layout filemerge -in layout.gds -in layout_block.gds \  
-out layout_design_with_block.gds -mode overwrite
```

Example 11 - Rename or Preserve Cells

In this example, any cells that have the same name are uniquely renamed, by adding a “_WBx” extension, where *x* is an integer. The result is an import-style merging of the two layouts.

```
layout filemerge -in layout.gds -in layout_block.gds \  
-out ./output/layout_design_with_block.gds -mode rename
```

Example 12 - Force Rename Cells

In this example, all cells are uniquely renamed, by adding a “_WBx” extension, where *x* is an integer. The result is a merge-style merging of the layouts.

```
layout filemerge -in layout.gds -in layout_block.gds \  
-out ./output/layout_design_with_block.gds -mode forcereaname
```

Example 13 - Use -smartdiff Argument to Compare Cells

In this example, -smartdiff is used to find the differences between two layouts. By default, the layout viewer compares cells by name alone, however to compare cell content as well you need to use the -smartdiff argument. After executing the layout filemerge command below, visually compare the two versions of the file-merged layout to identify the newer cells. If “-mode rename” is used without -smartdiff, identical cells with the same name are duplicated, however with -smartdiff only differing cells with the same name are duplicated.

```
layout filemerge -in dd1.gds -in dd2.gds -out outfile.gds \  
-smartdiff -mode rename
```

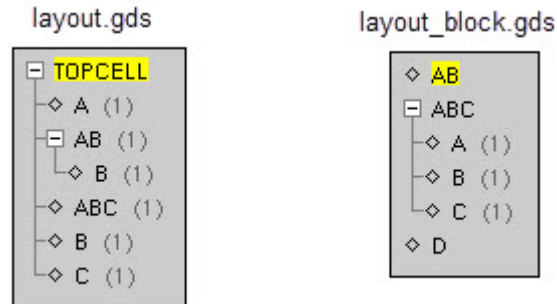
Example 14 - Increment (or Bump) Layers and Create a Topcell

This example combines the files “dd1.gds” and “dd2.gds” into the output file “outfile.gds.” All the layers in “dd1.gds” are incremented by 60. Any cells that have the same name are uniquely renamed. The result is a merging of the layouts, and the output file has a topcell named TOP.

```
layout filemerge -in dd1.gds 60 -in dd2.gds \  
-out outfile.gds -mode rename -createtop TOP
```

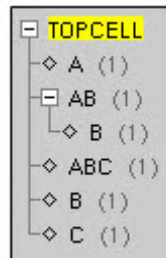
Example 15 - Specify a Topcell Name

This example uses the `-topcell` argument when merging `layout.gds` and `layout_block.gds` into `layout_with_block.gds`.

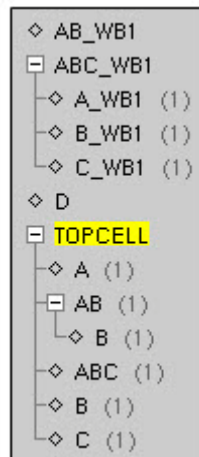


```
layout filemerge -in layout.gds -in layout_block.gds \  
-out ./output/layout_with_block_ntc.gds -mode rename \  
-topcell TOPCELL
```

The `-topcell` argument causes only the references in TOPCELL to be output to the resulting merged layout.



Without the `-topcell` argument, the cells from both layout databases are merged and the `-mode rename` argument renames any duplicate cellnames so that they are unique.



Example 16 - Allow Scaling

This example uses the `-scale` argument to scale *layout_block.gds*, merge with *layout.gds*, and output the results to *merged.oas*.

```
layout filemerge -in [list -name layout_block.gds -scale 2]
                  -in layout.gds -out merged.oas
```

Example 17 - Disable Strict Mode Output

This example merges the files *spare_top_fill.oas* and *spare_top_wIP.oas*. The `-strictmode` argument creates strict mode output for all fields in the table-offsets structure of the resulting *merged.oas* file. The `layout peek` command with the `-strictmode` argument returns a value of 1 indicating that the resulting file is in strict mode.

```
layout filemerge -in spare_top_fill.oas -in spare_top_wIP.oas \
                  -out ./output/merged.oas -strictmode 1
layout peek ./output/merged.oas -strictmode
1
```

Example 18 - Report Merge Conflicts

This example uses the `-reportconflicts` argument to identify any merge conflicts when merging the files *spare_top_fill.oas* and *spare_top_wIP.oas*. The merge mode defaults to `-mode append`, which concatenates the contents of the duplicate cells together.

```
layout filemerge -in spare_top_fill.oas -in spare_top_wIP.oas \
                  -out ./output/out.oas -reportconflicts
...
Note: cell spare_top already exists in spare_top_fill.oas, append with
spare_top from spare_top_wIP.oas.
```

Example 19 - Map Layers

In this example, the `-include_layer` and `-map_layer` arguments are applied globally to the input files *layout0.oas* and *layout1.oas*. Objects on layers 1, 3.789, and 99.99 in *layout0.oas* are appended to objects on the same layers in *layout1.oas*. Layer 1 is mapped to layer 10 and layer 3.789 is mapped to layer 4.654 in the merged results.

```
layout filemerge -in layout0.oas -in layout1.oas
                  -include_layer [list 1 3.789 99.99] \
                  -map_layer [list 1 10 3.789 4.654] \
                  -out ./output/merged.oas
```

Example 20 - Map Cells

This example uses the `-map_cell` argument to specify the name of the mapping file, *cell_map.txt*, that contains the following cell mappings:

```
a1240 layout1_a1240
a2311 layout1_a2311
a1220 layout1_a1220
a1620 layout1_a1620
a1720 layout1_a1720
```

The following layout filemerge command renames the cells, based on the mapping in *cell_map.txt*, prior to merging *layout0.gds* and *layout1.gds* into *merged.gds*.

```
layout filemerge -in layout0.gds -in layout1.gds -out merged.gds \  
-map_cell cell_map.txt
```

Example 21 - String Severity

This example uses the `-stringSeverity` argument to specify how an input OASIS file containing illegal a-string values is handled.

```
layout filemerge -in layout1.oas -in layout2.oas \  
-out ./output/merged.out -stringSeverity warning
```

The warning option for the `-stringSeverity` argument generates a message similar to the following for each illegal a-string value found in an input OASIS file and proceeds with merging the files.

```
Warning: at offset 12345 in file layout1.oas for cell 'TOPCELL', non-  
printable character 0xa in an a-string.
```

Example 22 - Specify an ASCII File as Input

This example uses the `-in textfile` argument set to specify an ASCII text file named *filemerge_inputs.txt* that contains the following input files for the layout filemerge operation:

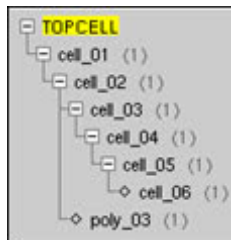
```
# this is a comment  
/home/user/design.gds  
/home/user/fill.oas -exclude_layer 4 -suffix _fill -exclude_top  
/home/user/standardcells/
```

Executing the following layout filemerge command merges *design.gds*, *fill.oas*, and any layouts found in the */home/user/standardcells/* directory. The file-specific arguments, `-exclude_layer`, `-suffix`, and `-exclude_top`, are applied only to the *fill.oas* layout.

```
layout filemerge -in filemergeinputs.txt -out output.oas
```

Example 23 - Empty Cell Handling

This example uses the `-noemptycells` argument to remove empty cell definitions and references to empty cells from the output file. In the input file (*input.gds*), the cells named *cell_03*, *cell_04*, *cell_05*, and *cell_06* are either empty or reference an empty cell. The cells named *cell_01*, *cell_02*, *poly_03*, and *TOPCELL* are not empty nor do they reference an empty cell.



The layout peek command with the -cells argument returns the following list of cells for the input file (*input.gds*):

```
layout peek input.gds -cells  
cell_05 cell_01 cell_06 cell_02 poly_03 cell_03 cell_04 TOPCELL
```

The layout filemerge command with the “-noemptycells 3” argument does not write empty cell definitions nor references to empty cells to the output file (*filter.gds*).

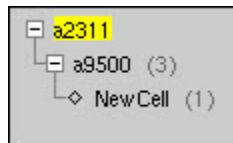
```
layout filemerge -in input.gds -out filter.gds -noemptycells 3
```

The layout peek command with the -cells argument returns the following list of cells for the output file, verifying that the empty cells and references to empty cells have not been written to the output file:

```
layout peek filter.gds -cells  
cell_01 cell_02 poly_03 TOPCELL
```

Example 24 - Excluding Cells

This example illustrates the results of using the -exclude_cell argument with the -deleteChildCells argument and the -noemptycells argument on a layout with the following hierarchy:



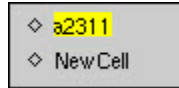
The following command suppresses the definition of a9500 from the merged results. However, references to a9500 are maintained but the cell is no longer defined. NewCell becomes a top cell since it was originally a child cell of a9500 and the -deleteChildCells argument was not used.

```
layout filemerge -in layout.gds -oasisout layout_with_childcell.oas \  
-exclude_cell a9500
```



The following command suppresses the definition of a9500 from the merged results. The -noemptycells argument causes all references to a9500 to be removed. NewCell becomes a top cell since it was originally a child cell of a9500 and the -deleteChildCells argument was not used.

```
layout filemerge -in layout.gds -oasisout layout_delete_childcell.oas \  
-exclude_cell a9500 -noemptycells 3
```



The following command suppresses the definition of a9500 from the merged results. The -deleteChildCells argument removes NewCell since it is a child of a9500 and was not referenced elsewhere in the hierarchy.

```
layout filemerge -in layout.gds -oasisout layout_nochildcell.oas \  
-exclude_cell a9500 -deleteChildCells
```



Related Topics

[overlay filemerge](#)

layout filtershapes

Removes polygon shapes from the layout file hierarchy.

Usage

layout filtershapes

```
-in filename
-filterfile filename
-maplayers '{
    '-filter {filterlayer | rulename}
    -filllayers filllayers
    [-auxlayers auxlayers]
    [-halo value]'...{'
-mapfile filename -out '{-outfile filename [-interactfile filename]}
[-topcell topcell]
[-halo value]
[-turbo {0 | 1}]
[-verbose {0 | 1}]
[-use_edges {0 | 1}]
[-deleteOrphanedCells {0 | 1}]
[-threads num_threads]
```

Arguments

- **-in *filename***
A required argument specifying an input layout file in GDS or OASIS format that contains fill polygons in the top-level cell.
- **-filterfile *filename***
A required argument specifying a filter file in GDS, OASIS, or RDB format. A filter file contains a marker layer or filter rule that identifies the fill polygons to remove from the input layout file.
- **-maplayers '{ ' '-filter {*filterlayer* | *rulename*} -filllayers '{ ' *filllayers* '}'**
[-auxlayers '{ ' *auxlayers* '}] [-halo *value*]'...{'
A required argument that maps a filter layer or filter rule in the file specified by **-filterfile** to the layers in the input layout file containing the polygons you want to filter. This argument is required only if the **-mapfile** argument is not specified. If **-mapfile** is specified in addition to this argument, the mappings defined in the mapfile are appended to the values specified by **-maplayers**.

The format for specifying a filter layer, fill layer, or auxiliary layer is:

layer [.datatype]

-filter {*filterlayer* | *rulename*} — Specifies a layer or rule in the filter file to be used for filtering polygons in the specified input layout file.

- filllayers** '{' *filllayers* '}' — Specifies one or more fill layers to filter in the input layout file. Braces with spaces are required when specifying more than one fill layer. The same fill layer can be specified for more than one filter layer or filter rule.
- auxlayers** '{' *auxlayers* '}' — Specifies one or more auxiliary layers to filter in the input layout file. Braces with spaces are required when specifying more than one auxiliary layer. The layers specified by the **-filllayers** argument are scanned for any overlapping polygons before the layers specified by **-auxlayers** are scanned.
- halo value** — Specifies a value (in microns) used to define a halo around the polygon shapes found on the filter layer or by the filter rule. If specified, the polygons found on the filter layer or by the filter rule are sized by an amount equal to the database precision multiplied by the specified halo value. This sizing occurs prior to finding any overlap with polygons on the fill layers of the input layout file. A filter-specific halo value takes precedence over a global halo value.

- **-mapfile filename**

A required argument used to specify the filename of a mapping file. The mapping file performs the same function as the **-maplayers** argument, mapping a filter layer or filter rule in the filter file to the layer numbers you want to filter in the input layout file. This argument is required if **-maplayers** is not specified. If **-maplayers** is also specified, the mappings in the mapfile are appended to the values specified by **-maplayers**. The syntax of this mapping file is:

```
-filter filterlayer | rulename -filllayers filllayers \  
[-auxlayers auxlayers] [-halo value]  
...
```

For example:

```
-filter 31.0 -filllayers {31.1 31.2} -auxlayers 51.0 -halo 0.1  
-filter 32.0 -filllayers {32.1 32.2} -auxlayers 52.0 -halo 0.1
```

- **-out** '{' **-outfile filename** [-interactfile filename] '}'

A required argument specifying the name of the output layout file. The format (GDS or OASIS) of the output file(s) is the same as the input file. You can use the syntax **-out filename** when specifying only one output file.

- outfile** — Use this option with the braces to specify multiple output files or when using the **-interactfile** option.
- interactfile filename** — Use this option to add to the top cell the overlapping polygon shapes that were removed from the input file and write the results to the specified file.

- **-topcell topcell**

An optional argument that allows you to specify the name of the top cell to use from the input layout. If not specified, the layout viewer automatically determines the top cell in the input layout file.

- **-halo *value***
An optional argument that specifies (in microns) a floating-point number that defines a halo that is applied globally to the polygon shapes found on the filter layers or defined by the filter rules. If specified, all polygon shapes on the filter layers or defined by the filter rules are sized by an amount equal to the database precision multiplied by the specified halo value before finding any overlap between polygons. A filter-specific halo value, if specified, takes precedence over a global halo value.
- **-turbo {0 | 1}**
An optional argument choice used to run layout filtershapes in single-threaded or multi-threaded mode. Valid options are:
 - 0 — Single-threaded mode.
 - 1 — Multi-threaded mode.
- **-verbose {0 | 1}**
An optional argument choice that reports additional warning messages to the console. Valid options are:
 - 0 — No additional reporting.
 - 1 — Reports additional warning messages to the console.
- **-use_edges {0 | 1}**
An optional argument choice used to convert edges in the specified filter file to polygons. This switch is only used if the filter file is in RDB format. Valid options are:
 - 0 — Edges are not converted to polygons.
 - 1 — Converts edges to polygons.
- **-deleteOrphanedCells {0 | 1}**
An optional argument choice that removes cells which are no longer referenced in the input layout hierarchy from the filtered output layout. This switch is enabled by default. Valid options are:
 - 0 — Cells that are no longer referenced after filtering become new top cells in the output file.
 - 1 — Cells that are no longer referenced after filtering are removed in the output file. This is the default.
- **-threads *num_threads***
Specifies the number of threads to use for running the filter operation. When this argument is not specified, the application uses the maximum number of CPUs that are available.

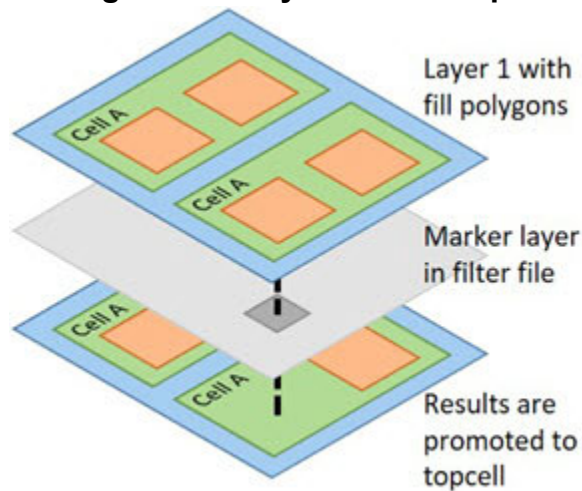
Return Values

None.

Description

Removes polygons from the hierarchy of the input layout file and outputs the results to a file. This command preserves the hierarchy of the input layout file. The input layout file must be GDSII or OASIS format. This command removes polygon shapes on the fill layer of the input layout that overlap with the polygon shapes in a filter file. [Figure 5-3](#) illustrates the basic concept of the layout filtershapes operation.

Figure 5-3. Layout Filtershapes



If auxiliary layers are specified, then shapes on the auxiliary layers that overlap with the shapes in the filter file are also removed. No cell can be modified except the top-level cell. Therefore, a cell reference from which an overlapping shape is removed, is flattened in the output file.

Property records are not preserved.

Examples

Example 1

In this example, *layout1.oas* is an input layout containing fill polygons in the top-level cell. A marker layer in the filter file, *layout2.oas*, contains polygons that identify the fill polygons to remove from the input layout.

```
layout filtershapes -in layout1.oas -filterfile layout2.oas \  
-mapfile map.txt -out ./output/output.oas
```

The map file, *map.txt*, specifies that layer 0 in the filter file is used to filter polygons on layer 1 in the input layout, and that the auxiliary layers, 0 and 3, are also filtered in the input layout after layer 1 is filtered:

```
-filter 0 -filllayers 1 -auxlayers {0 3}
```

The results are then written to the file *output.oas*.

Example 2

In this example, *layout0.gds* is an input layout containing fill polygons in the top-level cell. A filter rule in the filter file, *results.rdb*, contains polygons that identify the fill polygons to remove from the input layout. The `-maplayers` argument specifies that the `checkXYZ` rule name in the filter file is used to filter polygons on layer 2 in the input layout. The results are written to *filtered.rdb.gds* and the polygon shapes that were removed from the input layout are placed in the top cell and written to *interact.rdb.gds*.

```
layout filtershapes -in layout0.gds -filterfile results.rdb \  
-out {-outfile ./output/filtered.rdb.gds \  
-interactfile interact.rdb.gds} \  
-maplayers { { -filter checkXYZ -filllayers 2} }
```

Example 3

In this example, the `-use_edges` argument causes any edges in the filterfile to be converted to polygons. Fill polygons in the input layout that overlap with polygons in the filterfile are removed from the output file.

```
layout filtershapes -in layout.gds -filterfile filter_file.rdb \  
-maplayers { { -filter 0 -filllayers { 7 8 9 } } -use_edges 1 \  
-out ./output/layout_filtered.gds
```

Example 4

In this example, the `-auxlayers` argument specifies three layers (51, 52.70, and 53.70) in *XOR.oas* to filter in *layout.oas* once layer 31.0 has been filtered. The `-halo` argument defines a halo (0.3) that is applied to the shapes on the filter layer. The results are output to *results.oas* and the shapes that were removed from the input file are written to *filtered_items.oas*.

```
layout filtershapes -in layout.oas -filterfile XOR.oas -maplayers \  
{ { -filter 31.0 -filllayers { 31.2 31.3 } -halo 0.3 \  
-auxlayers { 51 52.70 53.70 } } -out { -outfile ./output/results.oas \  
-interactfile ./output/filtered_items.oas }
```

layout merge

Merges two layouts, with automatic renaming of the cells to prevent name conflicts.

Note



This command is not supported in HC mode.

Usage

```
layout merge [handle_out] {handle1 /file1} {handle2 /file2} bump [del1 del2] [-mode mode]  
[-preserveTextAttributes] [-preserveProperties] [-dt_expand] [-preservePaths]  
[-ignoreGdsBoxes] [-autoAlign]
```

Arguments

- *handle_out*
An optional argument specifying a handle to assign 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*
An optional argument set that controls whether the layouts used as input to the merge are deleted as part of the copy. Values for both arguments must be specified. The *del1* argument applies to the first layout to be merged, and *del2* applies to the second layout to be merged. Valid values for each argument include:
 - 1 — Deletes the input layout.
 - 0 — Does not delete the input layout. This is the default for each argument.
- -mode *mode*
An optional argument and *mode* defining how duplicate cells (cells from each input file that have the same name) are handled during the merge operation. Valid modes include:
 - append — Concatenates the contents from the duplicate cells into one cell with that name. All cells from both layouts with unique names appear in the final layout.
 - overwrite — Writes the contents of duplicate cells from the second file over the contents of the first, deleting the contents of the cell from the first file. All cells from both

layouts with unique names appear in the final layout. Cell definitions contain geometric data, while cell references do not.

rename — Renames the duplicate cell, to make it unique. All cells from both layouts with unique names appear in the final layout.

forcerename — Copies all cells from both layouts into the final layout as unique cells with unique names. The new name is created by adding a `_WB x` extension to the original name, where x is an integer. This is the default.

- **-preserveTextAttributes**

An optional argument instructing the tool to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the tool. Use this argument when merging a layout from a GDS file. The default is to preserve these attributes. This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- **-preserveProperties**

An optional argument used when merging a layout from a GDS file. Instructs the tool to preserve GDS geometry and reference property data (PROPATTR and PROPVALUE). The default is to preserve these properties. This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- **-dt_expand**

An optional argument to expand datatypes so that each layer and datatype combination is mapped to a different layout layer. The layer number within the layout is *layer.datatype*. This option is ignored when merging data from previously loaded layouts (using handles for both layouts).

- **-ignoreGdsBoxes**

An optional argument that instructs the tool to ignore box records when reading the GDS file.

- **-autoAlign**

An optional argument used only in Calibre MDPview. Calibre MDPview attempts to align two merged layout areas on top of each other where 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 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 the layout viewer accepts:

- **GDS files**


May have a variable database unit and user unit, and the measurement metrics are a ratio of database units to user unit. By default, if you create a new GDS file layout, 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**

Use the measurement item 'unit', which always has a size of 1 micron. When loading or creating an OASIS file, the layout viewer sets the database unit to be microns (1e-6 meters), with a user unit size of 1e6 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, the tool 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, 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 0.001 to 0.0005, and the layout to be scaled by 2. The scale changes an edge of 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
```

Return Values

The new layout handle.

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.

Examples

Example 1

This example merges the contents of two layouts, *layout2* and *layout4*, into the layout named *mhandle*. A value of 1000 is added to the layer numbers of each layer in *layout4* in order to prevent layer number conflicts. The default merge mode of *forcerename* is used, which copies the cells from both layouts into *mhandle* as unique cells with unique names.

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

Example 2

This example creates layouts from *file1.gds* and *file2.gds*, assigning the layouts to the variables L0 and L1, respectively. The layouts are then merged into a new layout using the layout merge command and the new layout is assigned to the variable M2. A bump value of 1000 is added to the layer numbers of each layer in *file2.gds* and duplicate cells are appended into one cell with the same name.

```
set L0 [layout create file1.gds -dt_expand]
set L1 [layout create file2.gds -dt_expand]
set M2 [layout merge $L0 $L1 1000 -mode append]
```

Related Topics

[layout filemerge](#)

layout overlays

Returns all defined overlay handles.

Usage

layout overlays

Arguments

None.

Return Values

Returns all defined overlay handles.

Description

Displays all defined overlay handles.

Examples

```
% layout overlays  
overlay0
```

Related Topics

[\\$O layoutHandle](#)

[\\$O layouts](#)

[\\$O overlayCells](#)

[\\$O overlayout](#)

[\\$L isOverlay](#)

[\\$L isReferenced](#)

[\\$L layerFilters](#)

layout peek

Returns information from a layout database without loading the file into memory.

Usage

layout peek *fileName* [-bbox [*cell1* [... *celln*]] [-geometryonly]] [-cache *directory*]
[-cblock | -cblockcount] [-cellcount] [-cellhlayers *cell1* [...*celln*]]
[-celllayers *cell1* [... *celln*]] [-cellproperty *cell1* [... *celln*]] [-cells]
[-celltext *cell1* [... *celln*]] [-child *cell1* [... *celln*]]
[-childbbboxes *parentcellname childcellname*] [-emptycells] [-fileproperty] [-format]
[-handle] [-lastmodtime] [-layers] [-maxblocksizes *number*] [-oasisCellIds]
[-parent *cell1* [...*celln*]] [-precision] [-refcount *cell1* [...*celln*]] [-reportprogress]
[-sproperty *property_name*] [-strictmode] [-topcell] [-topcells] [-undefcells] [-units]

Arguments

- ***fileName***

A required argument specifying the name of the layout database to be read. If it is not in the current directory, a path (relative or absolute) must be specified.

- -bbox [*cell1* [...*celln*]] [-geometryonly]

An optional argument that returns the bounding box of each specified cell. The bounding box for each cell is output as {*x y width height*}, where *x* and *y* represent the coordinates of the lower-left corner of the bounding box. The values of the bounding box are in dbu.

If no cell is specified, then the command returns the bounding box of the same topcell that would be returned by the -topcell argument. The output includes both the topcell name and the bounding box information.

Options to the -bbox argument include:

-geometryonly — An optional argument that specifies the -bbox argument should ignore text and empty cell references. The bounding box is output as {0 0 0 0} if the layout database does not contain any geometries for calculating the bounding box. This option is not supported with the -cache argument.

- -cache *directory*

An optional argument that creates or updates a PCR file. Executing the layout peek command with this option instructs the tool to locate the PCR file in the specified directory and update, as needed. If the PCR file is nonexistent, then it is created in the specified directory.

Note



The PCR file is always used when the -handle argument is specified. If the -cache *directory* information is not specified, the layout peek command uses the MGC_CWB_PCR_PATH variable. If this variable is not set, the layout peek command writes the cache file to the current directory.

- **-cblock | -cblockcount**

An optional argument that returns information about CBLOCKS in the layout database. The **-cblock** argument returns values based on the existence or non-existence of CBLOCKS in the layout database. Possible return values are 0 (CBLOCKS do not exist) and 1 (CBLOCKS exist). The **-cblockcount** argument returns the number of CBLOCKS in the layout database.

- **-cellcount**

An optional argument that returns the number of cells in the layout database.

- **-celllayers *cell1* [... *celln*]**

An optional argument that returns all layers present in the specified cell(s) and in the cell hierarchy.

- **-childbboxes *parentcellname* *childcellname***

An optional argument that returns the bounding box coordinates for each placement of the specified *childcellname* that is directly within the specified *parentcellname*. The bounding box coordinates are returned as {*x1 y1 x2 y2*}, where the first two values identify the lower left corner and the second two values identify the upper right corner of each bounding box. The coordinates are in database units (dbu). This option is not supported with PCR files.

- **-cells**

An optional argument that returns all of the cellnames that are in the layout database.

- **-child *cell1* [... *celln*]**

An optional argument that returns a list of child cells for the specified cell(s) that are in the layout database.

Note



Only the first level of child cells for the specified cell(s) are returned. Cells more than one level lower in the cell hierarchy are not returned.

- **-lastmodtime**

An optional argument that returns the date and time that the layout database was last modified and saved. This option is only supported with GDS files.

- **-layers**

An optional argument that returns all layers that are available in the layout database. Layers are returned as layer and datatype pairs.

- **-maxcblocksizes *number***

An optional argument that reports the largest CBLOCK sizes (in bytes) in the specified layout. The *number* option is a positive integer that specifies the number of the largest CBLOCKS in the layout to return. This argument returns the results in a Tcl list containing a pair of numbers, where the first number in the list is the uncompressed size and the second is the compressed size. The number of CBLOCKS returned is based on the number of occurrences of CBLOCKS in the layout, which may be less than the specified *number*, and is

sorted from largest CBLOCK to smallest. The default *number* is one when *number* is not specified, which returns only the largest CBLOCK size. This argument returns an empty list when no CBLOCKS are found.

- **-fileproperty**

An optional argument that returns a list of properties and associated property values that are in the layout database. This option is only valid for OASIS files and cannot be used with the **-cache** or **-handle** options.

- **-oasisCellIds**

An optional argument that reports (in a list format) the cell names and their associated cell IDs for an OASIS file. For layouts that are *not* in strict mode format, there may or may not be a cell ID for each cell that is referenced.

- **-parent *cell1* [... *celln*]**

An optional argument that returns a list of parent cells for the specified cell(s) that are in the layout database.

Note



Only the first level of parent cells for the specified cell(s) are returned. Cells more than one level higher in the cell hierarchy are not returned.

- **-celllayers *cell1* [... *celln*]**

An optional argument that returns all layers present only in the specified cell(s).

- **-celltext *cell1* [... *celln*]**

An optional argument that returns any text that is in the specified cell(s). It does not report text in any subcells. This option cannot be used with the **-handle** or **-cache** arguments.

- **-cellproperty *cell1* [... *celln*]**

An optional argument that returns the properties associated with the specified cell(s) in the layout database. This option is only valid for OASIS files and cannot be used with the **-handle** or the **-cache** option.

- **-precision**

An optional argument that returns the precision of the layout database. The precision is the ratio of database units to user units.

- **-refcount *cell1* [... *celln*]**

An optional argument that returns the number of references to the specified cell(s).


- **-topcell**

An optional argument that returns the name of the topcell. If there is more than one topcell, this argument returns the name of the topcell with the most descendants.

- **-topcells**

An optional argument that returns the names of all topcells in the layout database.

Note

 Some layout designs, usually those resulting from a merged database, have a structure containing more than one topcell. Any unreferenced cell is treated as a topcell.

- **-format**
An optional argument that returns the database format of the specified file.
- **-handle**
An optional argument used to assign a handle to the newly created peek object. Multiple layout peek queries can use the -handle option to avoid rereading layout files to extract the peek information. This is done by invoking the handle as “*handlename peek ...*”
- **-emptycells**
An optional argument that returns a list of cells that are empty. The list does not include undefined cells, which can be returned using the -undefcells argument.
- **-strictmode**
An optional argument that returns whether the OASIS file is in strict mode (OASIS-**STRICT-MODE**). The file is in strict mode if the table-offsets flag for all six table offsets is 1. This argument returns 0 (no strict mode) or 1 (strict mode).
- **-reportprogress**
An optional argument that displays status messages every few seconds to indicate the progress of the command. The status messages only display if the command takes more than a few seconds. A summary message displays in the Calibre DESIGNrev terminal window when the command completes and overwrites any progress messages.
- **-sproperty *property_name***
An optional argument that returns the values of OASIS standard properties present in the OASIS layout database. Each standard *property_name* has a value you can retrieve. The OASIS standard properties are listed in [Table 5-8](#):

Table 5-8. OASIS Standard Properties

Property Name	Value
File-Level Standard Properties	
S_MAX_SIGNED_INTEGER_WIDTH	Bytes to encode a signed integer.
S_MAX_UNSIGNED_INTEGER_WIDTH	Bytes to encode an unsigned integer.
S_MAX_STRING_LENGTH	Bytes allowed in any string.
S_POLYGON_MAX_VERTICES	Vertices allowed in any polygon.
S_PATH_MAX_VERTICES	Vertices allowed in any path.
S_TOP_CELL	Name(s) of the topcell(s) of a cell hierarchy.

Table 5-8. OASIS Standard Properties (cont.)

Property Name	Value
S_BOUNDING_BOX_AVAILABLE	Available bounding boxes. A value of 0 means bounding box properties are not provided. A value of 1 means some bounding box properties are provided. A value of 2 means there is a bounding box property for every CELLNAME record.
Cell-Level Standard Properties	
S_BOUNDING_BOX	Bounding box of the cell. The value consists of the following five fields: flags, lower left X, lower left Y, width, and height.
S_CELL_OFFSET	Byte offset from the beginning of the layout file to the corresponding CELL record. Note: A value of 0 indicates an external cell, a cell that is referenced but not defined in the layout.

Note

Refer to the *Open Artwork System Interchange Standard (OASIS)* for additional information concerning standard properties. The element level property S_GDS_PROPERTY is not supported by the layout viewers.

For the cell-level S_BOUNDING_BOX and S_CELL_OFFSET properties, you can also specify one or more cellnames, as these properties are associated with individual cells. For example:

- o -sproperty S_BOUNDING_BOX [*cellname1 cellname2 ...*]

This syntax is used to return the value of the S_BOUNDING_BOX property associated with a cell. If no cellname is specified, the tool returns the value of S_CELL_OFFSET for all cells. Otherwise, the result is the cellname and list of 5 bounding box values (flags, lower left X, lower left Y, width, and height). A warning is generated if the S_BOUNDING_BOX property is not present for the given cell or if the specified cellname is not present.

- o -sproperty S_CELL_OFFSET [*cellname1 cellname2 ...*]

This syntax is used to return the value of the S_CELL_OFFSET property associated with a cell. If no cellname is specified, the tool returns the value of S_CELL_OFFSET for all cells. A warning is generated if the S_CELL_OFFSET property is not present for a given cell or if the specified cellname is not present.

- **-undefcells**
An optional argument that returns the cells that are referenced but not defined in the layout database.
- **-units**
An optional argument that returns the database units (in microns) of the specified file.

Return Values

Returns information about the layout database without loading the layout into memory. The information returned is different for each argument:

- **-bbox** — Returns the coordinate space as $\{x\ y\ width\ height\}$ of the specified cells. 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.
- **-childbbboxes** — Returns a bounding box (lower-left and upper-right x, y coordinates) for each placement of the specified *childcellname* that is directly within the specified *parentcellname*.
- **-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 cell(s).
- **-lastmodtime** — Returns the creation date from the specified GDS file.
- **-layers** — Returns all layers available in the layout, including layers that are defined yet contain no shapes. For OASIS, the result for each layer contains the layer number, datatype number, and layername. For GDS, the result for each layer contains the layer number and datatype number.
- **-fileproperty** — Returns a list of properties and associated property values for the layout.
- **-parent** — Returns a list of parent cells for the specified cell(s).
- **-cellhlayers** — Returns all layers present in the cell and down its hierarchy.
- **-celllayers** — Returns all layers present in the cell.
- **-celltext** — For each text object found in a cell, the output includes the text string, coordinates, and layer.
- **-cellproperty** — Returns the properties associated with each specified cell.
- **-precision** — Returns the precision (1/user units) of the layout database.
- **-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.
- **-indexable** — Returns 1 if the OASIS file is indexable (for each CELLNAME record, the S_CELL_OFFSET property is present), otherwise the option returns 0.
- **-format** — Returns GDS or OASIS file type.
- **-strictmode** — Returns 1 if the OASIS file is in strict mode, otherwise this argument returns 0. The layout is considered to be in strict mode if the table-offsets flag for all six offset tables is 1.
- **-sproperty** — Returns the values of OASIS file-level and cell-level standard properties present in the layout database. Refer to [Table 5-8](#) for information on the properties and values that are returned.
- **-undefcells** — Returns a list of cell names that are referenced but not defined in the layout database.
- **-units** — Returns the units in microns.

Description

Displays information about a layout database, such as the units, precision, and list of cells, without loading the layout database into memory.

Note



The layout viewer tool returns an error if it cannot read the specified *fileName*. For example, to return information about a job deck, you must execute the **layout peek** command using **calibrewb** (Calibre WORKbench) or **calibremdpv** (Calibre MDPview).

This command returns only the precision of the layout database when *fileName* is the only argument specified. If more than one argument is specified, the values are returned in an array-assignable format. The following example shows an array-assignable format where the return values are delimited by the (highlighted) argument name.

```
% layout peek lab1a.oas -layers -cells -precision
precision 1000.0
cells {a1240 a2311 a1310 a1220 a1620 a9500 a1230 a1720 TOP}
layers {{2 0 2} {4 0 4} {5 0 5} {6 0 6} {7 0 7} {8 0 8} {9 0 9}
        {10 0 10} {1 0 1} {0 0 0} {300 0 300}}
```

Examples

Example 1 - Using Default Values

This example uses the default values for all arguments, which returns the database precision of *lab1a.gds*.

```
% layout peek lab1a.gds
1000.0
```

Example 2 - Using the -bbox Argument

This example uses the -bbox argument to return the bounding box for cells a1220 and a2311 in *lab1a.gds*. Specifying -bbox with no arguments returns the name and bounding box information for the topcell.

```
% layout peek lab1a.gds -bbox a1220 -bbox a2311
{a1220 {-4250 0 28500 85000}} {a2311 {-4250 0 52500 85000}}
```

Example 3 - Using the -cblock Argument

This example uses the -cblock argument to check *lab1a.oas* for the existence of CBLOCKS. A return value of 0 would indicate there are no CBLOCKS.

```
% layout peek lab1a.oas -cblock
1
```

Example 4 - Using the -cblockcount Argument

This example uses the -cblockcount argument to return the number of CBLOCKS found in *lab1a.oas*.

```
% layout peek lab1a.oas -cblockcount
8
```

Example 5 - Using the -childbbboxes Argument

This example uses the -childbbboxes argument to return the bounding box coordinates for the three placements of cell M1_P_27 that is in BUFX2.

```
% layout peek fullchip.oas -childbbboxes BUFX2 M1_P_27
BUFX2 {M1_P_27 {1250 2675 1430 3785} {865 2675 1045 3785}
      {490 2675 670 3785}}
```

Example 6 - Using the -cellcount Argument

This example uses the -cellcount argument to return the number of cells found in *fullchip.oas*.

```
% layout peek fullchip.oas -cellcount
295
```

Example 7 - Using the -cells and -precision Arguments

This example uses the -cells and -precision arguments to return the precision and the names of the cells found in *lab1a.gds*.

```
% layout peek lab1a.gds -cells -precision
precision 1000.0 cells {TOP a9500 a1310 a1240 a1230 a1720 a2311 a1220
a1620}
```

Example 8 - Using the -child Argument

This example uses the -child argument to return the children of the cells named BUFX2 and BUFX4.

```
% layout peek fullchip.oas -child BUFX2 BUFX4
{BUFX2 {M1_P_27 nmos_vt1_1 nmos_vt1_2 M1_POLY_26 pmos_vt1_3}}
{BUFX4 {M1_POLY_26 M1_P_27 nmos_vt1_1 nmos_vt1_0 pmos_vt1_3}}
```

Example 9 - Using the -lastmodtime Argument

This example uses the -lastmodtime argument to return the date and time that *lab1a.gds* was modified and saved.

```
% layout peek lab1a.gds -lastmodtime
2018/06/28 22:50:15
```

Example 10 - Using the -layers Argument

This example uses the -layers argument to return the layers in the layout database. Layers are output as a list of layer number and datatype pairs.

```
% layout peek lab1a.gds -layers
{10 0} {8 0} {0 0} {300 0} {9 0} {4 0} {7 0} {6 0} {5 0} {2 0} {1 0}
```

Example 11 - Using the -fileproperty Argument

This example uses the -fileproperty argument to return a list of properties and associated property values for the layout.

```
% layout peek layout.oas -fileproperty
{S_MAX_SIGNED_INTEGER_WIDTH 0} {S_BOUNDING_BOXES_AVAILABLE 0}
{S_MAX_STRING_LENGTH 0}
  {S_MAX_UNSIGNED_INTEGER_WIDTH 0} {S_PATH_MAX_VERTICES 0}
  {S_POLYGON_MAX_VERTICES 0} {S_TOP_CELL Nand}
```

Example 12 - Using the -parent Argument

This example uses the -parent argument to return the names of the parent cells of M1_POLY_26.

```
% layout peek fullchip.oas -parent M1_POLY_26
{M1_POLY_26 {BUFX4 DFFPOSX1 NOR3X1 NAND2X1 FAX1 OR2X1 OR2X2 OAI21X1
  XOR2X1 AOI21X1 NAND3X1 DFFSR INVX1 INVX2 INVX4 INVX8 XNOR2X1
  OAI22X1 AND2X1 AND2X2 AOI22X1 NOR2X1 HAX1 CLKBUF1 MUX2X1 BUFX2}}
```

Example 13 - Using the -cellhlayers and -celllayers Arguments

In this example, the -cellhlayers argument returns all layers present in the cell my_ram and its hierarchy, while the -celllayers argument returns the layers present only in the cell my_ram. Layers are returned as a list of layer number and datatype pairs.

```
% layout peek fullchip.oas -cellhlayers myram -celllayers myram
cellhlayers {myram {1 0} {3 0} {4 0} {5 0} {9 0} {10 0} {11 0} {12 0}
{13 0} {14 0} {15 0} {16 0} {17 0}} celllayers {myram {17 0} {15 0} {13 0}}
```

Example 14 - Using the -celltext Argument

This example uses the -celltext argument to return the top level text for the cell(s) found in the layout. Text information is returned as a list that includes the text string, coordinates, layer number, and datatype.

```
% layout peek fullchip.oas -celltext ADC_5bit_sc_5
{ADC_5bit_sc_5 {vdda 1890 186560 11 0} {gnd 1925 180920 11 0}
  {Vin 325 97700 13 0} {CLK 435 95985 13 0} {B0 56735 104330 13 0}
  {B3 56740 121260 13 0} {B4 56765 126905 13 0} {B1 56775 109995 13 0}
  {B2 56840 115605 13 0}}
```

Example 15 - Using the -cellproperty Argument

This example uses the -cellproperty argument to return a list of properties and associated property values for the cells named NewCell and TOPCELL.

```
% layout peek prop-cell.oas -cellproperty NewCell TOPCELL
{NewCell {{S_GDS_PROPERTY {1 propvalue}}}
{propname0 {prop0 prop1 prop2 prop3 prop4 prop5}}
  {propname1 {1 -1}}}} {TOPCELL {{propname1 1}
  {propname3 {1.0 1.5678}}}}
```

Example 16 - Using the -refcount Argument

This example uses the -refcount argument to return the number of references to the cell named M1_POLY_26.

```
% layout peek fullchip.oas -refcount M1_POLY_26
{M1_POLY_26 67360}
```

Example 17 - Using the -topcell Argument

This example uses the -topcell argument to return the name of the topcell with the most descendants.

```
% layout peek fullchip.oas -topcell
top
```

Example 18 - Using the -topcells Argument

This example uses the -topcells argument to return the names of all topcells in *fullchip_mod.oas*.

```
% layout peek fullchip_mod.oas -topcells
LV top NV RM myram_bdec myram_block AV myram_rdec
```

Example 19 - Using the -format Argument

This example uses the -format argument to return the format of the *fullchip.oas.gz* file.

```
layout peek fullchip.oas.gz -format
OASIS
```

Example 20 - Using the -emptycells Argument

This example returns a list of the empty cells found in *fullchip_empty_cells.oas*.

```
layout peek fullchip_empty_cells.oas -emptycells
NewCell1 NewCell2 NewCellXYZ NewCell
```

Example 21 - Using the -strictmode Argument

This example uses the -strictmode argument to return whether or not the OASIS file *lab1a.oas* is in strict mode.

```
% layout peek lab1a.oas -strictmode
1
```

Example 22 - Using the -sproperty Argument

This example uses the -sproperty argument to return the topcell name associated with the S_TOP_CELL OASIS property.

```
% layout peek input.oas -sproperty S_TOP_CELL
TOP
```

Example 23 - Using the -undefcells Argument

This Tcl script uses the -undefcells argument to report cells that are referenced but not defined in the layout *lab1a_TOP.gds*. The results are output to the text file *undefined_cells.txt*.

```
set outfile [open "./output/undefined_cells.txt" w]
puts $outfile [layout peek lab1a_TOP.gds -undefcells]
close $outfile
```

Example 24 - Using the -units Argument

This example uses the -units argument to return the user units used in *lab1a.gds*.

```
% layout peek lab1a.oas -units
0.001
```

Example 25 - Writing “layout peek” Results to a Text File

This Tcl script named *script.tcl* outputs the results of the layout peek command using the -cells and -precision arguments to the file *results.txt*.

```
set outfile [open "./output/results.txt" w]
puts $outfile [layout peek mix.gds -cells -precision]
close $outfile
```

Enter the following to execute the script:

```
% calibredrv script.tcl
```

Example 26 - Using the -reportprogress Argument

This example uses the -reportprogress argument to report the progress of the layout peek command when using the -maxblocksize argument.

```
% layout peek layout.oas -maxblocksize -reportprogress
```

Progress messages are output to the Calibre DESIGNrev terminal window during the operation. For example:

```
Read 43 MB out of 505 MB (43.0 MB/s)

Read 119 MB out of 505 MB (59.5 MB/s)

Read 219 MB out of 505 MB (54.8 MB/s)

Read 241 MB out of 505 MB (60.2 MB/s)

Read 505 MB in 7 seconds (72.1 MB/s)
```

Example 27 - Reporting CBLOCK Sizes

This example uses the `-maxcblocksizes` argument to report the five largest CBLOCKS in *layout.oas*.

```
% layout peek layout.oas -maxcblocksizes 5
{639445926 233576884} {114567 54663} {49963 12440} {47782 13258} {45321
10131}
```

Related Topics

[\\$P delete](#)

[\\$P file](#)


[\\$P peek](#)

[\\$L holdupdates](#)

layout tbmode

Specifies the text box extent treatment behavior.

Note

 The extent argument is not supported in HC mode.

Usage

layout tbmode [origin | extent]

Arguments

- origin | extent

An optional argument specifying the mode to be used.


origin — This mode 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. This is the default.

extent — This mode sets the text extent to be database driven by the height of the text object. 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.

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


Note

 This argument is not supported in HC mode.

Return Values

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

Note

 This command returns the value of the current global tbmode. It does not return the value of the tbmode for any existing layout objects in memory.

Description

Specifies how text extents are handled. Once a layout is loaded or created, the tbmode for that layout is fixed.

Setting the tbmode with this command affects only text box attributes for layout objects yet to be loaded. The command returns the new handling mode, though it does not exhibit in the currently loaded design. After you load a new cell, it has text box attributes of the last layout tbmode command.

If a particular text box attribute is desired, the command should be executed before loading the cell. For example:

```
layout tbmode extent
set L1 [layout create layout0.gds]
```

Examples

This example sets the text extent to the lower-left and upper-right coordinate:

```
layout tbmode origin
```

Related Topics

[\\$L bbox](#)

Layout Object Methods

Use the layout object methods to operate on objects in a layout, such as cells, polygons, references, and layers. The layout object methods described in this section can be used for all layout viewers.

A layout object is referenced by its handle, which also acts as a command for operations performed on the layout. The command reference pages in this section use the \$L variable to refer to the layout handle. You must assign a value to this variable name when executing the command. You can use `$cwb cget - _layout` to return the handle of the layout currently visible in the layout viewer main window.

Table 5-9. Layout Object Methods Summary

Layout Object Commands	Description
<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 a list of all the cells in the layout \$L.
<code>\$L children</code>	Returns unique children (but not subsequent descendants) of the specified cell.
<code>\$L clips</code>	Returns clips that are defined for the specified cell.
<code>\$L clipsout</code>	Write the currently-loaded clips to a file.
<code>\$L connect</code>	Defines connectivity between 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>	Creates a clip of the specified cell.
<code>\$L create layer</code>	Creates a new layer with the specified layer number.
<code>\$L create polygon</code>	Creates a polygon with the given coordinates.
<code>\$L create ref</code>	Creates a placement of a cell reference.
<code>\$L create text</code>	Creates a text object in the specified cell.
<code>\$L create wire</code>	Creates a wire (also called a path) on <i>layer</i> in <i>cellName</i> with given coordinates.
<code>\$L customLayerDrawOrder</code>	Changes the layer drawing order to the specified ordering.

Table 5-9. Layout Object Methods Summary (cont.)

Layout Object Commands	Description
\$L delete cell	Deletes the specified cell and all references to that cell.
\$L delete clip	Deletes the specified clip.
\$L delete duplicate object	Deletes duplicate objects (references, paths, polys, and texts) from the specified cell(s).
\$L delete duplicate ref	Deletes the specified duplicate array and cell references (AREF or SREF).
\$L delete layer	Deletes the specified layer.
\$L delete objects	Deletes objects from the specified cell and layer.
\$L delete polygon	Deletes the polygon described by its layer and coordinates from the specified cell.
\$L delete polygons	Deletes all geometries from the specified layer of a specified cell. The layer is left in the Layer table.
\$L delete ref	Deletes the cell reference or array of references. Cell references do not contain geometric data.
\$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 exportNet	Exports nets from a specified cell and layer to a file.
\$L exportView	Exports a specified view of a layout to a file.
\$L extractNet	Performs geometry-based or property-based net extraction.
\$L file	Returns the filename for the current 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 a cell reference.
\$L format	Queries the layout file format.
\$L gdsout	Writes the layout to a GDS-formatted file.

Table 5-9. Layout Object Methods Summary (cont.)

Layout Object Commands	Description
\$L gridsnap	Depending on the arguments, either snaps the geometries and cell references to the specified value of <i>gridsize</i> OR checks for violations of a design grid based on <i>gridsize</i> .
\$L holdupdates	Pauses cell bounding box calculation.
\$L import layout	Imports the specified layout. Options define the manner in which cellname 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	Determines if there have been any modifications made to the layout since the last save was performed.
\$L isOverlay	Determines if the layout is an overlay.
\$L isReferenced	Determines if the layout is referenced in one or more overlays.
\$L iterator (poly wire text)	Returns a list of the specified type of objects.
\$L iterator (ref sref aref)	Returns a list of the specified type of references.
\$L iterator count (poly wire text)	Returns the number of the specified type of objects.
\$L iterator count (ref sref aref)	Returns the number of the specified type of references.
\$L layerconfigure	Configures a layer with the specified layer properties.
\$L layerFilters	Defines a layer filter that can be used to filter the layers shown in the Layers Browser.
\$L layernames	Returns a list of layer numbers and layernames in the layout, or assigns a new layer name to the specified layer number.
\$L layers	Returns a list of all layer numbers in this layout.
\$L layoutType	Returns or sets the layout file type.
\$L libname	Sets or gets the GDS LIBNAME variable.
\$L loadlayerprops	Loads layer properties from the specified layer property file.
\$L MASK_LAYER_INFO	Returns the area of shapes on a layer within a specified cell. Also returns the lower-left coordinates of the layer's extent, along with the width and height of the extent bounding box.
\$L maxdepth	Returns the maximum depth of the specified cell in the layout.
\$L modify layer	Changes an existing layer to a new layer or an existing layer and datatype pair to a new layer and datatype pair.

Table 5-9. Layout Object Methods Summary (cont.)

Layout Object Commands	Description
\$L modify origin	Changes the origin of a specified cell.
\$L modify text	Modifies the layer number, coordinates, text string, presentation (GDS layout only), and text properties of an existing text object.
\$L NOT	Performs a BOOLEAN NOT on the specified layers.
\$L oasisout	Writes the layout to an OASIS-formatted file.
\$L options	Checks to see the options that were set when the layout was created (using layout create).
\$L OR	Performs a BOOLEAN OR on the specified layers.
\$L property	Globally replaces a property in a GDS or OASIS file, or adds a file-level property to an OASIS file.
\$L pushmarkers	Pushes the error markers from the top level cell of an ASCII Results Database (RDB) to the cells in which the errors occur. The criteria for pushing markers can be based on an exact match of error markers or a mapping of polygons to error markers.
\$L query	Returns information about an object of the specified type in a specified cell, level of hierarchy, and location.
\$L rdbout	Converts all polygons on the specified layers into RDB output.
\$L readonly	Sets the read-only state for a layout.
\$L refcount	Reports the number of instances of a specified cell or a total count of the instances in the specified parent cell.
\$L report duplicate ref	Outputs duplicate cell reference information about a layout.
\$L ruler	Imports or exports rulers from a specified cell in the current layout.
\$L scale	Scales the layout by the given value.
\$L SIZE	Sizes the specified layer.
\$L SNAP	Snaps the data in a layer onto a new layer.
\$L srefsFromAref	Returns a list of SREFs converted from an AREF string.
\$L stopconnect	Defines stopping layer(s) for a net connection layer, or returns all possible input or stopping layer combinations.
\$L textout	Displays all text elements in the given cell as a string.
\$L topcell	Returns the topcell name.
\$L transcript	Turns recording of edit transcript information on or off.

Table 5-9. Layout Object Methods Summary (cont.)

Layout Object Commands	Description
\$L transcript output	Writes the edit transcript to an output file.
\$L transcript range	Returns a range of edit transcript lines.
\$L transcript state	Returns the edit transcript state (whether recording is on or off).
\$L units database	Sets or returns the size of database unit in meters.
\$L units microns	Sets or returns the number of database units per micron.
\$L units user	Sets or returns the ratio of user units per database units.
\$L update	Updates the layout by recalculating the bounding boxes, topological ordering, and layer tables.
\$L viacell	Specifies the named cell is a via cell.
\$L XOR	Performs a BOOLEAN XOR on the specified layers.

\$L allowupdates

Resumes cell bounding box calculations.

Usage

\$L allowupdates

Arguments

None.

Return Values

None.

Description

Provides a direct way to resume cell bounding box (bbox) calculation updates earlier or on request. This command can only be called from within a Tcl procedure, and not from the command line.

Bbox calculations are 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 executed, it automatically calls [\\$L allowupdates](#) at the end of a Tcl procedure.

Caution



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

Examples

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 updated.

The `create_layout_and_add_shapes_sample2` procedure suspends bbox calculations until the call to [\\$L allowupdates](#). 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 Topics

[\\$L holdupdates](#)

\$L ancestors

Returns unique ancestors for the specified cell.

Usage

\$L ancestors *cellName* [-level *value*]

Arguments

- *cellName*
A required argument specifying the name of the cell whose ancestors are to be returned.
- -level *value*
An optional argument used to limit the output to those ancestors found by going recursively to the specified level. The integer specified for *value* must be greater than 0. For example, specifying a level of 1 returns unique ancestors of the specified cell, specifying a level of 2 returns unique ancestors to the second level of the hierarchy, and so on.

Return Values

Unique ancestors of the specified cell and, optionally, to the specified level.

Description

Returns the names of unique ancestors for the specified cell. An ancestor is defined as any cell that has the specified cell in its subhierarchy. For example, if cell A references cell B which in turn references cell C, then executing this command with a *cellName* value of C returns both A and B.

Examples

Example 1

This example script opens the layout named *fullchip.oas* and outputs the ancestors of the cell named “CV” to *results.txt*.

```
set lay [layout create fullchip.oas]
set results [open ./output/results.txt w]
puts $results [$lay ancestors CV]
close $results
exit
```

Example 2

This example script opens the layout named *fullchip.oas* and uses the -level argument to return the unique ancestors up to the first level of the hierarchy for the cell named “M1_POLY_55”. The results are output to *results.txt*.

```
set cwb [find objects -class cwbWorkBench]
set lay [layout create fullchip.oas]
set results [open ./output/results.txt w]
puts $results [$lay ancestors M1_POLY_55 -level 1]
close $results
exit
```

Related Topics

[\\$L children](#)

\$L AND

Performs a Boolean AND on the specified layers.

Usage

\$L AND *InputLayer1 InputLayer2 OutputLayer* [-cell *cell* [-hier {0|1}]]

Arguments

- ***InputLayer1***
A required argument specifying the name of the first layer to AND with the specified second layer.
- ***InputLayer2***
A required argument specifying the name of the second layer to AND with the specified first layer.
- ***OutputLayer***
A required argument specifying the name of the output layer for the AND operation. The output layer specified should not be an input layer.
- -cell *cell*
An optional argument used to specify the name of a cell in which to perform a Boolean AND operation. The AND operation is performed on the current cell unless this argument is specified. An error is returned if the specified *cell* does not exist or is empty.
- -hier {0|1}
An optional argument used to enable or disable performing the Boolean AND operation across all levels of the hierarchy for the specified *cell*. The default (0) disables performing the operation across the hierarchy. When enabled, this argument flattens the portion of the design (based on the specified *cell*) before performing the boolean operation.

Return Values

None.

Description

Performs a Boolean AND on the specified layers and merges the output to the layer ***OutputLayer***. The operation can be performed on the current cell or a specified cell.

Examples

Example 1

This example ANDs layers 2 and 4, outputs the results to layer 100, and then updates the GUI display.

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

Example 2

This example flattens all levels of the hierarchy in cell a1240, ANDs layers 5 and 8, outputs the results to layer 300, and then updates the display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
%lay AND 5 8 300 -cell a1240 -hier 1
% $wb updateDisplay
```

Related Topics[\\$L NOT](#)[\\$L OR](#)[\\$L XOR](#)

\$L asciiout

Writes the results in the layout to an ASCII file.

Usage

\$L asciiout *fileName*

Arguments

- *fileName*

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

Return Values

None.

Description

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

- Layernames are written as rule check names.
- All text objects are written as rule check text lines.
- If more than one cell is present, it results 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.

Examples

Example 1

This example creates a layout from a results database and then uses the [\\$L layoutType](#) command to set the file type to ASCII. The **\$L asciiout** command writes the results to the ASCII file named *my_results.asc*.

```
set lay [layout create my_results.rdb]
$lay layoutType set ascii
$lay asciiout ./output/my_results.asc
```

Example 2

This example creates a layout from an OASIS file and then uses the [\\$L layoutType](#) command to set the file type to ASCII. The **\$L asciiout** command writes the results to the ASCII file named *my_results.asc*.

```
set lay [layout create my_layout.oas]
$lay layoutType set ascii
$lay asciiout ./output/my_results.asc
```

Related Topics

[\\$L instancedbout](#)

\$L bbox

Returns the bounding box of the specified cell.

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 instructs the tool to recompute and update the bounding box for the specified cell.

Note



This option is not affected by [layout tbmode](#) commands executed after the layout is loaded.

- **-s_bounding_box**
An optional argument that instructs the tool to retrieve the S_BOUNDING_BOX property value (for layout viewer compatibility with the OASIS.MASK standard). This option may only be specified for OASIS-format files.
- **-p44_chip_window**
A required argument instructing the tool to retrieve the P44_CHIP_WINDOW property value (for layout viewer compatibility with the OASIS.MASK standard). This option may only be specified for OASIS format files and is required when ***cellName*** is not specified.

Return Values

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

Description

Displays the bounding box of the specified cell, in database units. The bounding box includes text and extents of the cell placements. The first two values are the lower left corner coordinates and the last two values are the upper right corner coordinates of the bounding box. The bounding box of an empty cell is (0, 0, 0, 0).

Examples

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

Related Topics

[layout tbmode](#)

\$L cellname

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.

Return Values

None.

Examples

Example 1

This example renames a cell:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay cellname a1620 a1620r
% $wb updateDisplay
```

Example 2

This example renames all cells in a layout:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% set mycells [$lay cells]
% foreach cell $mycells {
    set newname "n20_$cell"
    $lay cellname $cell $newname
}
```

Related Topics

[\\$L cells](#)

[\\$L children](#)

\$L cells

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

Usage

\$L cells

Arguments

None.

Return Values

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

Examples

Example 1

This example displays a list of all the cells in a layout:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay cells
lab1 a1220 a9500 a1720 a1230 a1310
```

Example 2

This example finds the layers used in each of the cells in a layout:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% set mycells [$lay cells]
lab2 a1500 a1890 a1900 a1960 a2010
% foreach cell $mycells {
    puts "Layers used in cell $cell are: [$L layers -cell $cell]"
}
Layers used in cell lab2 are: 6 2
Layers used in cell a1500 are: 4 2 1
Layers used in cell a1890 are: 8 6 4 2
Layers used in cell a1900 are: 8 2 1
Layers used in cell a1960 are: 8 6 4 2
Layers used in cell a2010 are: 8 6 4
```

Related Topics

[\\$L cellname](#)

[\\$L children](#)

[\\$L layers](#)

\$L children

Returns unique children (but not subsequent descendants) of the specified cell.

Usage

\$L children *cellName*

Arguments

- *cellName*

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

Return Values

Returns unique children, but not subsequent descendants, of the specified cell.

Description

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

Examples

Example 1

This example finds the children of a cell:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay cells
lab4 a1000 a2000 a3000
% $lay children lab4
a1000 a2000 a3000
% $lay children a2000
```

Example 2

This example finds the topcell and its children:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% set topc [$lay topcell]
lab7
% $lay children $topc
lab7 a1120 a1500 a1700 a1830 a1910
```

Related Topics

[\\$L ancestors](#)

[\\$L cellname](#)

[\\$L cells](#)
[\\$L topcell](#)

\$L clips

Returns clips that are defined for the specified cell.

Usage

\$L clips *cellName*

Arguments

- *cellName*

A required argument specifying the name of a cell.

Return Values

A Tcl list of the attributes for each clip found in the specified cell. The attributes output for each clip include:

- Clip ID.
- Clip name.
- Clip coordinates, starting with the lower left X and Y and followed by the upper right X and Y.
- Boolean stating whether a marker dot is placed in the center of the clip when the clip is drawn.
- Boolean stating whether the area of the clip is incrementally loaded.

For example:

```
{1 {Clip 1} 1716 52549 16921 20622 true false}
```

Description

Returns information about the clips that are defined for the specified cell. Clips are rectangular regions (identified by four coordinates) that you define in a layout. Clip information is independent of the layout.

Examples

Example 1

In this example, the Tcl script opens the layout named *lab.gds* and loads the clips file named *lab.clips*. The \$L clips command outputs the results to the *clips.txt* file.

Tcl script:

```
set lay [layout create lab.gds]
set top [$lay topcell]
set results [open ./output/clips.txt w]
$lay create clip top -clipsFile lab.clips
puts $results [$lay clips top]
close $results
exit
```

lab.clips:

```
<?xml version="1.0" standalone="yes"?>
<!-- Clips are managed by the application. -->
<clips>
  <version>1.0</version>
  <units>micron</units>
  <clip>
    <name>Clip 1</name>
    <x>-1.836</x>
    <y>277.303</y>
    <width>236.901</width>
    <height>251.593</height>
    <cellname>top</cellname>
  </clip>
  <clip>
    <name>Clip 2</name>
    <x>413.200</x>
    <y>290.158</y>
    <width>260.775</width>
    <height>235.065</height>
    <cellname>top</cellname>
  </clip>
  <clip>
    <name>Clip 3</name>
    <x>870.475</x>
    <y>277.303</y>
    <width>121.205</width>
    <height>247.920</height>
    <cellname>top</cellname>
  </clip>
</clips>
```

clips.txt:

```
{1 {Clip 1} -1836 277303 235065 528896 false false} \
{2 {Clip 2} 413200 290158 673975 525223 false false} \
{3 {Clip 3} 870475 277303 991680 525223 false false}
```

Related Topics

[\\$L clipsout](#)

[\\$L create clip](#)

[\\$L delete clip](#)

\$L clipsout

Write the currently-loaded clips to a file.

Usage

\$L clipsout *fileName cellName*

Arguments

- *fileName*
A required argument specifying the name of the generated file.
- *cellName*
A required argument specifying the name of the cell.

Return Values

None.

Description

Writes the currently loaded clip markers to a file that is loaded with the \$L create clip command in a separate, interactive instance of the layout viewer.

Examples

Example 1

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
```

Example 2

The following example script creates two clip markers, one with a zero width and height, and the other with non-zero width and height (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 how script2.tcl could be used:

```
$ calibredrv -shell script2.tcl  
$ calibredrv lab2.gds -incr lab2.clips
```

Related Topics

[\\$L clips](#)

[\\$L create clip](#)

[\\$L delete clip](#)

\$L connect

Defines connectivity between layers.

Usage

\$L connect *layer1 layer2* [{by *layer3*} | {via *cellname*}]

Arguments

- ***layer1***
A required argument specifying the first original layer or a group of original layers to connect.
- ***layer2***
A required argument specifying a second original layer or group of original layers to connect.
- **by *layer3***
An optional argument specifying a third original layer by which to connect ***layer1*** and ***layer2*** objects.
- **via *cellname***
An optional argument set specifying the name of a via cell. This argument creates a connection between ***layer1*** and ***layer2*** objects by way of a cell reference (*cellname*).

Return Values

Returns existing connection definitions for a specific layer. In this case, the command returns layers connected to the specified layer. To distinguish a via layer, a non-via layer always lists itself in the list of layers it connects to.

Note



Duplicate layer numbers are possible.

Description

Defines connectivity between objects on the specified input layers. Only original layout layers can be used with this command. The layout viewer does not perform Boolean operations or device extraction. Layer names are case sensitive.

If layernames are specified, all layers with the same name are used as input layers. Wildcards can be used when specifying layer numbers and datatypes. For example, if the layerprops file contains the following:


```
1 blue diagonal_2 M0 1 1
1.1 blue diagonal_left_wide M0 1 1
2 purple wave_small VIA 1 1
2.1 purple wave VIA 1 1
3 red diagonal_1 M1 1 1
3.1 red diagonal_right_wide M1 1 1
```

Then the following \$L connect statement would connect layers 1, 1.1, 3, and 3.1 by layers 2 and 2.1.

```
$L connect 1.* 3.* by 2.*
```

Once layer connectivity is defined, you can use the [\\$L extractNet](#) batch command to perform a geometry-based or property-based net extraction.

Caution

 The \$L connect and [\\$L disconnect](#) commands ignore ordering of the *layer1* and *layer2* layers. For example, the following commands are considered identical and result in only one connection: “\$L connect 1 2” and “\$L connect 2 1”.

The \$L connect command cannot be used on overlays.

Examples

Example 1

This example connects layers 0 and 2.40 by layer 1. It then uses the \$L extractNet command to perform a geometry-based net extraction starting with a polygon located on layer 0 in TOPCELL and at the x, y coordinates 500, 500. The \$L rdbout command outputs the results to an RDB file named *nets.rdb*.

```
set L [layout create layout0.gds -dt_expand]
$L connect 0 2.40 by 1
$L extractNet TOPCELL -geom 0 500 500
$L rdbout ./output/nets.rdb -cell TOPCELL -layers 2.40 1.0 0.0
```

Example 2

In this example, the layerprops file uses the same layername for multiple layers. For example:

```
1 blue diagonal_2 M0 1 1
1.1 blue diagonal_left_wide M0 1 1
1.2 blue alt_speckle M0 1 1
2 purple wave_small VIA 1 1
2.1 purple wave VIA 1 1
3 red diagonal_1 M1 1 1
3.1 red diagonal_right_wide M1 1 1
3.2 red alt_speckle M1 1 1
```

The following script uses the \$L connect command to define the connectivity between layers. The results of the \$L connect operations are output to *nets.gds* using the \$L extractNet command.

```
set L [layout create my_layout.gds -dt_expand]
set wb [find objects -class cwbWorkBench]
set topcell [$L topcell]
$L loadlayerprops my_layout.gds.layerprops
$L connect M0 M1 by VIA
$L connect M1 0
$L extractNet TOPCELL -geom 1 -1000 -13000
$wb updateDisplay
$L gdsout ./output/nets.gds
```

Example 3

This example uses the \$L connect command to connect layers 11 and 13 by the via cells whose name is prefaced by “top_VIA*”. The connections are then extracted to an OASIS file named *nets.oas*.

```
set L [layout create top_ic2.oas -dt_expand]
set wb [find objects -class cwbWorkBench]
set topcell [$L topcell]
$L connect 11 13 via top_VIA*
$L extractNet top -geom 11 500000 701500
$wb updateDisplay
$L oasisout ./output/nets.oas
```

Related Topics

[\\$L disconnect](#)

[\\$L extractNet](#)

\$L COPY

Copies the specified 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**. If the layer does not exist, the tool creates it.


Return Values

None.

Description

Copies the specified layer to the output layer. The copy operation is performed on a cell-by-cell basis.

Tip

 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.

Examples

This example copies layer 2 to layer 200:

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

Related Topics

[\\$L COPYCELL GEOM](#)

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

\$L COPYCELL GEOM

Copies the geometries on a given layer in the specified cell.

Note



This command is not supported in HC mode.

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 name of the cell into which the geometry is copied.
- ***destLayer***
A required argument specifying the layer number or *layer.datatype_number* designation for the layer into which the geometry is copied.

Return Values

None.

Description

Copies all geometries on a specific layer in the cell of a source layout into the specified cell and layer of the current layout. This command is not a hierarchical operation.

Examples

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

Related Topics

[\\$L COPY](#)

\$L create cell

Creates a new cell with the specified name.

Usage

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

Arguments

- *cellName*
A required argument specifying the name of the cell to be created. The name cannot be an existing cell name.
- *handle2*
An optional argument specifying the handle for the layout that contains *cell2*, whose contents are copied to the new cell.
- *cell2*
An optional argument specifying the name of the cell whose contents are copied to the new cell.

Return Values

None.

Description

Creates a new cell with the given name. By default, this method creates an empty cell. Supplying the optional arguments *handle2* and *cell2* 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. It creates the named cell, and then copies all geometries (no placements are copied) into the cell.

Examples

Extracts a 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 Topics

[\\$L create layer](#)

[\\$L create polygon](#)

[\\$L create ref](#)

[\\$L create text](#)

[\\$L create wire](#)

\$L create clip

Creates a clip of the specified cell.

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 an existing cell to use when creating the clip. The clip that is created is associated with the specified cell.
- **-layer {*layer* | *}**
An optional argument specifying the layers to display in the clip. You can specify one layer or specify a wildcard (*) for all layers. By default, the statement uses all visible layers in the layout viewer.
- **-depth *depth***
An optional argument specifying the hierarchy depth setting to use for the clip. By default, the statement uses the current depth setting in the layout viewer. When not in GUI mode, the tool uses depth 0.
- **-name *name***
An optional argument specifying the name of the clip. The default is “clip #”.
- **-incr**
An optional argument specifying to incrementally load the area of the clip, if the layout is incrementally loaded.
- **-mark**
An optional argument specifying to place a marker in the center of the clip.
- **-halo *microns***
An optional argument specifying to create a halo around polygons, in microns. By default, the statement creates no halos.
- **-clip {*llx lly urx ury*}**
An optional argument specifying two pair of X and Y coordinates that define the lower left and upper right corners of the bounding box for the clip. Coordinates should be enclosed in brackets with the Tcl list command (see “[Example 6 - Using the -name, -mark, and -clip Arguments](#)”). By default, when coordinates are not specified the command uses the specified cell to define the coordinates of the bounding box for the clip. The default coordinates are in database units (dbu).

Coordinates are 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*

An optional argument specifying the name of a file containing clip information to load. The file can be an existing clips file or an ASCII file containing coordinates for the clip(s).

Note



While not required, the recommended naming convention is to use the *.clips* suffix for a clips file and the *.coords* suffix for a coordinates file.

When specifying the name of an existing clips file, the specified **cellname** must exist in both the layout and in the clips file.

The following rules apply to an ASCII file containing coordinates:

- Each line must be a set of coordinates or an empty line. Empty lines are ignored.
- Each set of coordinates must contain either two or three coordinate values. The first coordinate value is X, the second coordinate value is Y, and if specified, the third value is the width and height of the clip. The default width and height is 1 micron if a third value is not specified.
- The coordinate values must be either real numbers representing microns or integer values representing database units (dbu).
- The coordinate values can be suffixed with u (micron) or d (dbu). Micron (u) is the default when a suffix is not specified.
- Comments can be included and must begin with the “#” character and end with a newline.
- Any lines containing text that is not a comment or a valid set of coordinates is treated as an invalid line.

When reading a coordinates file, the \$L create clips command creates a clip for each set of valid coordinates. The X and Y coordinate values are used for the clip names that appear in the layout viewer GUI (expressed in microns) and enclosed in parenthesis. You can use the [Clips palette](#) in the GUI to view and save the clips to a *.clips* file.

The \$L create clip command generates an error message and, in some cases, a summary message when the coordinates file:

- Contains no valid lines or valid coordinates. The error message indicates the coordinates file is an invalid file format and the command fails.
- Contains valid lines in addition to one or more invalid lines. The command generates an explicit error message for each invalid line (up to five errors). A summary message is generated when there are more than five errors.

Refer to “[Example 8 - Creating Clips from a Valid Coordinates File](#)” on page 269 and “[Example 9 - Creating Clips from an Invalid Coordinates File](#)” on page 269 for examples of valid and invalid coordinates file.

- `-layout {handle | *} {cellName | *} layer`

An optional argument specifying the layout handle, cell, and layer to use when creating the clip.

- `-rdb rdbFile {ruleCheck | *}`

An optional argument specifying the name of an RDB file and rule check to use when creating the clip. You can specify a wildcard (*) to indicate that all rule checks in the RDB file should be used to create the clip(s). The error in the RDB file determines the bounding box

Return Values

Returns the ID of the newly-created clip.

Description

Creates clips that are associated with the specified cell in a layout. The bounding box of the clip can be based on the specified cell (default), a set of coordinates (-clip), an existing clips file or a file containing a set of coordinates (-clipsFile), a layout (-layout), or an RDB file (-rdb).

You can use the \$L clipsout command to write the currently loaded clips to a file.

Examples

Example 1 - Using Default Values

This example creates a clip of the cell named “lab1”. The [\\$L clipsout](#) command outputs the clip results of cell “lab1” to a clips file named *lab1.clips*.

```
set lay [layout create lab1.gds]
$lay create clip lab1
$lay clipsout ./output/lab1.clips lab1
```

Example 2 - Using the -layer Argument

This example creates a clip using layer 8 from cell a1220.

```
$lay create clip a1220 -layer 8
```

Example 3 - Using the -layer Argument

This example creates a clip using all layers from cell a1220.

```
$lay create clip a1220 -layer *
```

Example 4 - Using the -name Argument

This example creates a clip of the cell named “a1620” and assigns the name “my clip” to the clip. The **\$L clipsout** command outputs the results of cell “a1620” to the clips file named *design.clips*.

```
set lay [layout create design.gds]
$lay create clip a1620 -name "my clip"
$lay clipsout ./output/design.clips a1620
```

Example 5 - Using the -clipsFile Argument

This example creates a clip from the clips file named *design.clips* and outputs the results to the file *clips_out.clips*.

```
set lay [layout create design.gds]
set topcell [$lay topcell]
$lay create clip lab1 -clipsFile design.clips
$lay clipsout ../output/clips_out.clips $topcell
```

Example 6 - Using the -name, -mark, and -clip Arguments

The following script creates two clips: the first set of coordinates creates a clip having zero width and height, while the second set of coordinates creates a clip having width and height (perhaps showing a region to inspect). The script then writes the clips to the file *lab1.clips* that can be opened from the **Clips** tab in the layout viewer.

```
layout create lab1.gds
set lay [lindex [layout all] 0]
set topcell [$lay topcell]
$lay create clip $topcell -name "Clip 1" -mark -clip [list 10000.0 \
  200000.0 10000.0 200000.0] \
$lay create clip $topcell -name "Clip 2" -mark -clip [list 10000.0 \
  300000.0 140000.0 400000.0]
$lay clipsout ../output/lab1.clips $topcell
```

Example 7 - Using the -rdb and -halo Arguments

The following example creates clips from all of the results in the RDB file named *lab1.drc.results*. A bounding box for each clip is created with a 2um halo around the result. The script then writes the clips to the file *lab1.clips* that can be opened from the **Clips** tab in the layout viewer.

```
layout create lab1a.gds
set lay [lindex [layout all] 0]
set topcell [$lay topcell]
$lay create clip lab1 -rdb lab1.drc.results * -halo 2
$lay clipsout ../output/lab1.clips $topcell
```

Example 8 - Creating Clips from a Valid Coordinates File

This example uses the following coordinates file (*clips.coords*) to create clips using the \$L create clip command:

```
#Coordinates for first clip
96.152 361.398 180
#Coordinates for second clip specified in database units
343478d 343478d 144281d
#Coordinates for third clip specified in micros
101.125u 222.145u
#Coordinates for fourth clip specified in database units and microns
100.u 347580d 450.5u
```

The script creates four clips in *lab.gds* using the coordinates specified in *clips.coords* and outputs the clips to *lab.clips* that can be opened from the **Clips** tab in the layout viewer.

```
layout create lab.gds
set lay [lindex [layout all] 0]
set topcell [$lay topcell]
$lay create clip top -clipsFile clips.coords
$lay ./output/clipsout lab.clips $topcell
```

Example 9 - Creating Clips from an Invalid Coordinates File

In this example, the following script uses the \$L create clip command to load a clips file containing invalid characters named *clips.coords*. The error is output to *results.txt*.

```
set lay [layout create lab.gds]
set topcell [$lay topcell]
set clipsfile clips.coords
set results [open ./output/results.txt w]
if {[catch {$lay create clip $topcell -clipsFile $clipsfile} errMsg]} {
    puts $results "The clips file named '$clipsfile' contains \
    invalid characters."
    puts $results " "
    puts $results "$errMsg"
}
close $results
```

clips.coords:

```
# Invalid characters highlighted in orange
X 12350 width
553702d Y 653245d
553.702 653.245d -125.467
472470d 192304d 132623d Height
553702 653.245d 125.467
$@@@
```

results.txt:

```
The clips file named 'clips.coords' contains invalid characters.
```

```
Usage: "layout0 create clip ... -clipsFile <file>"; invalid file format
```

Related Topics

[\\$L clips](#)

[\\$L clipsout](#)

[\\$L delete clip](#)

\$L create layer

Creates a new layer with the specified layer number.

Usage

\$L create layer *L*

Arguments

- *L*

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

- An integer in the range of 0...65535.
- A pair of integers separated by a period (.). This format is used to represent a layer and datatype pair as *layer.datatype*.

Return Values

None.

Examples

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

Related Topics

[\\$L create cell](#)

[\\$L create polygon](#)

[\\$L create ref](#)

[\\$L create text](#)

[\\$L create wire](#)

\$L create polygon

Creates a polygon with the given coordinates.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

Usage

\$L create polygon *cellName layer* {[-prop *attr string* [G | U]]} ... *x1 y1 x2 y2 ... xn yn*

Arguments

- ***cellName***
A required argument specifying the name of the cell in which the polygon is 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 argument plus associated values defining polygon property data (PROPATTR and PROPVALUE). More than one -prop argument can be specified.

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

G (GDS) is the default and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.
- ***x1 y1 x2 y2 ... xn yn***
A required argument set specifying the coordinates of the vertices of the polygon. If only two vertices are supplied, the tool interprets the polygon as a box. At least two coordinates are required. By default, coordinates are in database units (dbu).

Coordinates are 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.

Return Values

None.

Description

Creates a polygon on the specified layer and cell using the specified 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.

Examples

Example 1

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}
```

Example 2

This example creates a polygon a2200 on layer 420:

```
% 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 Topics

[\\$L create cell](#)

[\\$L create layer](#)

[\\$L create ref](#)

[\\$L create text](#)

[\\$L create wire](#)

[\\$L delete polygon](#)

\$L create ref

Creates a placement of a cell reference.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

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 in which to place the 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 reference. By default, coordinates are in database units (dbu).

Coordinates are 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. Valid values include:
 - 0 — Do not apply a mirror.
 - 1 — Mirror across the x axis of the cell.
The mirror is applied to the cell reference prior to any rotation specified by ***angle***.
- ***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 degrees. The value specified for ***mirror*** is applied to the cell reference prior to any rotation specified by ***angle***.
- ***mag***
A required argument specifying the magnification to apply to the cell reference. The default is 1, which equates to no magnification. The value of mag can be any real number greater than 0.

- *cols rows xspace yspace*

An optional argument set that specifies the location of a cell reference. The array must have at least two rows or two columns. The *cols* and *rows* values define the number columns and rows in the array, while *xspace* and *yspace* define the horizontal and vertical spacing between the origins of cells in the adjacent columns and rows. These values can be negative to reflect relative positions below or to the left of the array origin.

Creating an array that has more than 65535 columns or rows generates an error. The limit for coordinate space is 32-bit; an error is generated if the *xspace* or *yspace* values exceed this limit.

For example, specifying the following:

```
$L create ref cellA cellB 1000000 1000000 0 0 1 65536 0 100 100
```

Generates the following error:

```
Error: <layout> create ref: aref cols argument must be no greater than 65535
```

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

An optional argument used to assign a property attribute (PROPATTR) and associated value (PROPVALUE) to the cell reference. Valid options include:

attr — The name of the property attribute (PROPATTR).

string — The value (PROPVALUE) to assign to the property attribute.

G — Specifies the property is a GDS property. For OASIS databases, this value specifies the property is a standard OASIS property that represents a GDS-style property. This is the default and is the only allowed option for GDS databases.

U — Specifies the property is an OASIS user property. If you write a file containing OASIS user properties to a GDS file, the OASIS properties are discarded.

- *-force*

An optional argument that forces the creation of an empty cell reference if the specified cell reference (*refcell*) does not exist. The bounding box of an empty cell is (0, 0, 0, 0).

Return Values

Returns coordinates of the cell reference placement.

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 and magnification of the cell reference. Optional arguments can be used to create an array of cell references.

Examples

Example 1

This example creates a cell reference and then queries for the cell reference.

```
% set cwb [find objects -class cwbWorkBench]
% 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 {}}
```

Example 2

This example creates a cell reference of the cell named a1720. The reference is placed at 0, 0 (x y coordinates), mirrored across the x axis, and rotated 90 degrees. There is no magnification applied to the cell reference. The **\$L query** command then queries for the cell reference and outputs the results to a text file.

```
set lay [layout create lab1a.gds]
set top [$lay topcell]
$lay create ref $top a1720 0 0 1 90 1
set file [open ./output/results.txt w]
puts $file "$lay query ref $top 0 9 0 0"
close $file
```

Example 3

This script creates a new layout and populates it with data from two other layouts. It creates references to cells in the new layout, creates a new layer, creates a polygon on the new layer, and outputs the results to a GDS file.

```
# 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

# Create cell ONE and TWO in the new layout. Copy the contents of the
# topcell in Lone to cell ONE and the contents of the topcell in Ltwo
# to cell TWO.
# Copy patterns to new layout
$Lnew create cell ONE $Lone topcell
$Lnew create cell TWO $Ltwo topcell

# Create cell references to cell ONE and TWO and place in the new layout.
$Lnew create ref TOP ONE 0 0 0 0 1
$Lnew create ref TOP TWO 0 6000 0 0 1

# Create a new layer in the new layout if it does not exist.
set newlayer 10
set predicate [$Lnew exists layer $newlayer]
if {$predicate == 0} {
    $Lnew create layer $newlayer
}

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

# Write the results to a GDS file.
$Lnew gdsout new.gds
```

Related Topics

[\\$L create cell](#)

[\\$L create layer](#)

[\\$L create polygon](#)

[\\$L create text](#)

[\\$L create wire](#)

[\\$L delete ref](#)

\$L create text

Creates a text object in the specified cell.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

Usage

\$L create text *cellName layer x y 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 is created.
- ***layer***
A required argument used to define the layer number or *layer_number.datatype_number* designation for the layer containing the text object.
- ***x y***
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 are 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 text string.
- ***presentation* [*strans magnification angle*]**
An optional text attribute value, 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 argument 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.

attr — An integer that is the attribute number (PROPATTR).

string — An ASCII string that is the value (PROPVALUE) associated with the property attribute (PROPATTR).

G — The property is a GDS property (default). When used with an OASIS database, this option specifies the property is a standard OASIS property that represents a GDS-style property.

U — The property is an OASIS user property. This option cannot be specified for GDS databases. OASIS user properties are discarded when writing to a GDS file.

- `-yoffset yoff`

An optional argument specifying the text offset in database units. If *text* contains any newline characters, each new text object is offset in increments of *yoff*.

Return Values

None.

Examples

Example 1

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
```

Example 2

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

```
layout0 create text TOPCELL 4 0 0 "Sample Text" 0 0 50 270
```

Example 3

This example adds a cellname as text in a cell:

```
set cwb [find objects -class cwbWorkBench]
set L [$cwb cget -_layout]
set celllist [layout peek my_layout.gds -child my_cellname]

set cells0 [regsub "{" $celllist "" cells1]
set cells2 [regsub "}" $cells1 "" cells3]
set cells [split $cells3]

set a 0
foreach cellnametext $cells {
    set cellnametextnum [lindex $cells $a]
    $L create text $cellnametextnum 127 0 0 $cellnametextnum
    incr a
}
```

Related Topics

[\\$L create cell](#)

[\\$L create layer](#)

[\\$L create polygon](#)

[\\$L create ref](#)

[\\$L create wire](#)

[\\$L delete text](#)

\$L create wire

Creates a wire (also called a path) on *layer* in *cellName* with given coordinates.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

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 wire is created.
- *layer*
A required argument specifying the layer number or *layer_number.datatype_number* designation for the layer on which to create the wire.
- *width*
A required argument specifying the width of the path to create. By default, width is in database units (dbu).
A coordinate length is 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. Valid values include:
 - 0 — Path with no extension at the beginning or end. This is the default.
 - 2 — Path with beginning and end extensions equal to 1/2 the path width.
 - 4 — Custom path, with user supplied values for the beginning and end extensions.
- *bgnextn*
An optional argument specifying the length of the beginning extension for the path. Required when the path type is 4 (a custom path). This argument cannot be specified with a path type of 0.
- *endextn*
An optional argument specifying the length of the beginning extension for the path. Required when the path type is 4 (a custom path). This argument cannot be specified with a path type of 0.

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

An optional argument plus associated values defining polygon property data (PROPATTR and PROPVALUE).

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

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.

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

A required argument set specifying the coordinates of the vertices of the wire. At least two sets of coordinates are required. If only two vertices are supplied, the tool interprets the polygon as a rectangle. By default, coordinates are in database units (dbu).

Coordinates are 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.

Return Values

None.

Description

Calibre DESIGNrev has a limit of 8192 points for polygons, so wires (or paths) with more than 4090 vertices are automatically segmented into multiple polygons. When this occurs, Calibre DESIGNrev issues a warning similar to the following:

```
WARNING: New wire on layer 33.0 of cell "mw_cell" contains 4191 vertices
(limit is 4090), converting to polygon.
```

Examples

Example 1

This example creates a path with given coordinates:

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

Example 2

This example creates a path and adds a property.

```
layout0 create wire TOPCELL 1 2 -prop 1 VDD G 28922 -62831 28922 -62671
```

Example 3

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 Topics

[\\$L create cell](#)

[\\$L create layer](#)

[\\$L create polygon](#)

[\\$L create ref](#)

[\\$L create text](#)

[\\$L delete wire](#)

\$L customLayerDrawOrder

Changes the layer drawing order to the specified ordering.

Usage

\$L customLayerDrawOrder [*layer_list*]

Arguments

- *layer_list*

An optional argument used to specify the layer drawing order. The new layer order along with each layer number must be separated by spaces. If this optional argument is not specified, the command displays the current layer number ordering.

Return Values

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

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.

Examples

This example changes the layer drawing order. The ordering is not 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
```

\$L delete cell

Deletes the specified cell and all references to that cell.

Usage

\$L delete cell *cellname* [-deleteChildCells]

Arguments

- *cellname*
A required argument specifying the name of the cell to delete.
- -deleteChildCells
An optional argument that deletes any child cells that are referenced in the specific cell's hierarchy. Child cells referenced outside of the hierarchy of the specified cell are not deleted.

Return Values

None.

Examples

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

Related Topics

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete text](#)

[\\$L delete wire](#)

\$L delete clip

Deletes the specified clip.

Usage

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

Arguments

- *cellName*
A required argument specifying the name of the cell to delete clips from.
- -id *id*
An optional argument specifying the identifier of a particular clip.

Return Values

None.

Examples

```
% 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 Topics

[\\$L clips](#)

[\\$L clipsout](#)

[\\$L create clip](#)

\$L delete duplicate object

Deletes duplicate objects (references, paths, polys, and texts) from the specified cell(s).

Note



This command is only supported in HC mode.

Usage

\$L delete duplicate object *cell* [... *cell_n*] [-out *filename*]

Arguments

- *cell* [... *cell_n*]
A required argument specifying one or more cells in which to remove duplicate references, paths, polygons, and texts. You can use the wildcard character '*' to find zero or more cell occurrences. For example, using the wildcard in "ca*t" finds cat and cart.
- -out *filename*
An optional argument specifying the name of a file to output the results of this operation. Specifying "stdout" outputs the report to the transcript.

Return Values

None.

Examples

Example 1

```
% layout0 delete duplicate object inv
```

\$L delete duplicate ref

Deletes the specified duplicate array and cell references (AREF or SREF).

Usage

\$L delete duplicate ref *cell_0* [... *cell_n*] [-useProps] [-out *filename*]

Arguments

- *cell_0* [... *cell_n*]
A required argument specifying one or more cells in which to remove duplicate references. The wildcard character ‘*’ is supported, and it finds zero or more cell occurrences. For example, using a wildcard in a search for ca*t finds cat and cart.
- -useProps
An optional argument specifying to consider property values in addition to other parameters of the reference.
- -out *filename*
An optional argument specifying the name of a file to output the results of this operation. Specifying “stdout” outputs the report to the transcript.

Return Values

None.

Examples

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

Related Topics

[\\$L delete cell](#)

[\\$L delete ref](#)

[\\$L report duplicate ref](#)

\$L delete layer

Deletes the specified layer.

Usage

\$L delete layer *L*

Arguments

- *L*

A required argument specifying the layer number or *layer_number.datatype_number* designation of the layer to delete.

Return Values

None.

Examples

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

Related Topics

[\\$L delete cell](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete text](#)

[\\$L delete wire](#)

\$L delete objects

Deletes objects from the specified cell and layer.

Usage

\$L delete objects *cell layer* [-polygons] [-wires] [-texts]

Arguments

- *cell*
A required argument specifying the name of the cell in which to delete the specified object types.
- *layer*
A required argument specifying the layer number and optional datatype number (for example, 10.2) on which to delete the specified object types.
- -polygons
An optional argument used to delete all polygons on the specified layer at the top-level in the specified cell.
- -wires
An optional argument used to delete all wires (paths) on the specified layer at the top-level in the specified cell.
- -texts
An optional argument used to delete all texts on the specified layer at the top-level in the specified cell.

Return Values

None.

Description

This command deletes objects (polygon, wires, and texts) from the specified cell and layer. You can specify multiple object types for deletion. Specifying no object types performs the same function as specifying all object types.

Examples

Example 1

This example deletes all polygons on layer 4 in topcell.

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

Example 2

This example deletes all polygons, wires, and texts on layer 10 in topcell.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay delete objects topcell 10
```

Related Topics

[\\$L delete polygon](#)

[\\$L delete text](#)

[\\$L delete polygons](#)

[\\$L delete wire](#)

\$L delete polygon

Deletes the polygon described by its layer and coordinates from the specified cell.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

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 argument plus associated values defining polygon property data (PROPATTR and PROPVALUE).

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

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.
- ***x1 y1 x2 y2 ... x_n y_n***
A required argument specifying 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 are 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.

Return Values

None.

Examples

Example 1

This 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
```

Example 2

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 Topics

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete text](#)

[\\$L delete wire](#)

\$L delete polygons

Deletes all geometries from the specified layer of a specified cell. The layer is left in the Layer table.

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 polygon(s) to be deleted.

Return Values

None.

Examples

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

Related Topics

[\\$L delete cell](#)
[\\$L delete layer](#)
[\\$L delete polygon](#)
[\\$L delete ref](#)
[\\$L delete text](#)
[\\$L delete wire](#)

\$L delete ref

Deletes the cell reference or array of references. Cell references do not contain geometric data.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

Usage

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

Arguments

- **incell**
A required argument specifying the name of the cell containing the cell reference to be deleted.
- **refcell**
A required argument specifying the name of the cell reference to be deleted.
- **x y**
A required argument specifying the coordinates of the cell reference to be deleted. By default, coordinates are in database units (dbu).
Coordinates are 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 applied to the cell reference to be deleted. Valid values include:
 - 0 — Do not apply a mirror.
 - 1 — Mirror across the x axis of the cell.
- **angle**
A required argument specifying the angle of rotation applied to the cell reference.
- **mag**
A required argument specifying the magnification applied to the cell reference.
- *cols rows xspace yspace*
An optional argument set that specifies the location of the cell reference. Refer to “[\\$L create ref](#)” on page 274 for more information.

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

An optional argument plus associated values defining polygon property data (PROPATTR and PROPVALUE) that must match property data in the specified *refcell* in order for the reference to be deleted.

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

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.

Return Values

None.

Examples

Example 1

This example deletes a cell reference:

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

Example 2

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 Topics

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete text](#)

[\\$L delete wire](#)

\$L delete text

Deletes the specified text.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

Usage

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

Arguments

- **cellName**
A required argument specifying the name of the cell containing the text object to delete.
- **layer**
A required argument specifying the layer number or *layer_number.datatype_number* designation for the layer containing the text object.
- **x1 y1**
A required set of arguments specifying the coordinates of the text object to be deleted. By default, coordinates are in database units (dbu).
Coordinates are 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 to be deleted.
- **presentation strans magnification angle**
An optional argument 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 \(GDS or OASIS file\)](#) 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 argument 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 G for the third argument indicates that this is a standard OASIS property that represents a GDS-style property. Use U to indicate an OASIS user property.

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.

Return Values

None.

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 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.
- In order 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.

Examples

Example 1

The following example deletes the text “X1”:

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

Example 2

This example script deletes all text in at the topcell 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 { $text_num > 0 } {
    # Find if layer does have some text labels
    for {set j 0} {$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 $topcell $lay $text_x $text_y $text_str
      } else {
        $L delete text $topcell $lay $text_x $text_y $text_str \
          [lindex $inst 3] [lindex $inst 4] [lindex $inst 5]
          [lindex $inst 6]
      }
    }
  }
}
$L gdsout "output.gds"
```

Example 3

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 Topics

[\\$L create text](#)

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

[\\$L delete ref](#)

[\\$L delete wire](#)

\$L delete wire

Deletes the specified path.

Note



For scripts that include a large number of \$L create or \$L delete commands, it is recommended that you use the [\\$L holdupdates](#) command to improve the performance.

Usage

\$L delete wire *cellName layer width* [-pathtype *type* [*bgnextn endextn*]]
[-propattr *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 is 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 are 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 delete. Allowed values include:
 - 0 — A path with no extension at the beginning or end (default).
 - 2 — A path with beginning and end extensions equal to 1/2 the path width.
 - 4 — A custom path, with user supplied values for the beginning and end extensions.
- ***bgnextn***
An optional argument specifying the length of the beginning extension for the path. This argument is required when the path type is 4.
- ***endextn***
An optional argument specifying the length of the beginning extension for the path. This argument is required when the path type is 4.
- **-prop attr string [*G* | *U*]**
An optional argument plus associated values defining polygon property data (PROPATTR and PROPVALUE).

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

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are 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 database units (dbu).

Coordinates are 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.

Return Values

None.

Description

Deletes the wire (also called a path) defined by the specified properties. (*layer* in *cellName* with given width and coordinates.)

Examples

Example 1

This example creates a path:

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

Example 2

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 Topics

[\\$L delete cell](#)

[\\$L delete layer](#)

[\\$L delete polygon](#)

[\\$L delete polygons](#)

\$L delete ref

\$L exists cell

\$L disconnect

Disconnects connectivity layers.

Usage

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

Arguments

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

Return Values

None


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 Calibre DESIGNrev and restarting the tool.

Internally, layernames are translated into layer numbers, so if layernames are used to create \$L disconnect statements, a query on \$L disconnect commands returns layer numbers. Layernames must be loaded before using them in a \$L disconnect statement.

This command is not supported on overlays.

Caution

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

```
$L disconnect 1 2
$L disconnect 2 1
```

Examples

```
% 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 Topics

[\\$L connect](#)

\$L duplicate cell

Duplicates a cell.

Usage

\$L duplicate cell *cellname*

Arguments

- *cellname*

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

Return Values

Name of duplicated cell.

Description

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

Examples

```
layout0 duplicate cell c1250
```

Related Topics

[\\$L create cell](#)

[\\$L delete cell](#)

\$L exists cell

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

Usage

\$L exists cell *cellName*

Arguments

- *cellName*

A required argument specifying the name of the cell to check for.

Return Values

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

Examples

```
% 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 Topics

[\\$L exists layer](#)

\$L exists layer

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

Usage

\$L exists layer *layer_num*

Arguments

- *layer_num*

A required argument specifying the layer number or *layer_number.datatype_number* designation for the function to check for.

Return Values

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

Examples

Example 1

This example checks if various layers exist:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay exists layer 10
1
% $lay exists layer 14.4
1
% $lay exists layer 123456789
0
```

Example 2

This script sets a variable to “1” if a layer exists:

```
set L [layout create pattern.gds -dt_expand]
set exP [$L exists layer 12.7]
puts $exP
```

Related Topics

[\\$L exists cell](#)

[\\$L isLayerEmpty](#)

\$L expand cell

Flattens all references of the specified cell.

Usage

\$L expand cell *cellName*

Arguments

- *cellName*

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

Return Values

None.

Description

Replaces all references to a specified cell with the contents of that cell. This brings all geometries in the cell reference up one level of hierarchy, however references to cells within the cell are left intact, and the cell definition is unaffected.

The cell definition itself is left intact and is not deleted from the design. Cell definitions contain geometric data, while cell references do not.

Note



Expanding a cell specifies the cell whose instance 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.

Examples

The following example expands the cell named “b”. Before expanding the cell, “topcell” contains an instance of cell b and cell a also contains an instance of cell b. Expanding cell b results in the contents of cell b becoming part of topcell as well as part of cell a:

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

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

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

Related Topics

[\\$L expand ref](#)

[\\$L flatten cell](#)

\$L expand ref

Flattens the specified cell references.

Usage

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

Arguments

- **incell**
A required argument specifying the cell containing the reference.
- **refcell**
A required argument specifying the name of the cell being referenced.
- **x y**
A required argument specifying the coordinates of the cell reference to be expanded. By default, coordinates are in database units (dbu).

Coordinates are 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 applied to the cell reference to be expanded. Valid values include:
 - 0 — Do not apply a mirror.
 - 1 — Mirror across the x axis of the cell.
- **angle**
A required argument specifying the angle of rotation applied to the cell reference.
- **mag**
A required argument specifying the magnification applied to the cell reference.
- *cols rows xspace yspace*
An optional argument set that specifies the location of the cell reference. Refer to “[\\$L create ref](#)” on page 274 for more information.
- -prop *attr string* [*G* | *U*]
An optional argument plus associated values defining polygon property data (PROPATTR and PROPVALUE).

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

G (GDS) is the default, and is the only allowed option for GDS files. If you write a file containing OASIS user properties (U) as a GDS formatted file, the OASIS properties are discarded.

Return Values

None.

Description

Replaces the specified reference(s) or array of references with the contents of a cell. It brings all geometries in the cell up one level of hierarchy, and references to cells within the reference are left intact.

Expand a cell reference 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.

Note



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 is not updated by future changes to that cell. Cell references do not contain geometric data.

Examples

The following example expands a reference:

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

Related Topics

[\\$L expand cell](#)

[\\$L flatten ref](#)

\$L exportNet

Exports nets from a specified cell and layer to a file.

Usage

\$L exportNet *cell* -layer *layer* { {-rdb|-gds|-oasis} *file* }

Arguments

- *cell*
A required argument specifying the name of the cell to export.
- *layer*
A required argument specifying the layer to export.
- {-rdb|-gds|-oasis} *file*
A required argument choice specifying the type and name of the database in which to export the nets.

Description

The \$L exportNet command provides the ability to export nets from a specified cell and layer to a file.

Examples

Example 1

In this example, connectivity is defined between layers using the \$L connect command and then extracted using the \$L extractNet command. The nets are then exported to an OASIS file using the \$L exportNet command.

```
set L [layout create layout.oas]
set results "./output/netsout.oas"
$L connect 15 17 by 16
catch {$L extractNet [$L topcell] -geom 17 17000 15000} reply
set net_layer $reply
puts "Net: $net_layer"
set layer [lindex $net_layer 0]
$L exportNet [$L topcell] -layer $layer -oasis $results
```

Related Topics

[\\$L connect](#)

[\\$L extractNet](#)

\$L exportView

Exports a specified view of a layout to a file.

Usage

```
$L exportView {-format output_format} {-filename path} {-cellname cell_name}  
  {-layers list_of_layers} {-width x_value} {-height y_value} {-depth depth}  
  {-clip clip_coordinates}
```

Arguments

- **-format *output_format***
A required argument used to specify the format of the output. PNG is currently the only supported output format.
- **-filename *path***
A required argument used to specify the name of the file in which to export the view.
- **-cellname *cell_name***
A required argument used to specify the name of the cell to export.
- **-layers *list_of_layers***
A required argument used to specify one or more layer numbers to export. If more than one layer is specified, the layers must be specified as a Tcl list. For example, using the Tcl list command:

```
-layers [list 2 3 4 6]
```
- **-width *x_value***
A required argument used to specify the width (in pixels) of the output. The value specified for *x_value* must be a positive integer.
- **-height *y_value***
A required argument used to specify the height (in pixels) of the output. The value specified for *y_value* must be a positive integer.
- **-depth *depth***
A required argument used to specify the maximum depth of the cell hierarchy to export.
- **-clip *clip_coordinates***
A required argument used to specify two pairs of x- and y-coordinates (in user units) that define the lower left and upper right corners of the area to export. The coordinates must be specified as a four element Tcl list of integers or floating point numbers. For example, using the Tcl list command:

```
-clip [list 120 350 520 500]
```

Description

This command exports a view of a layout to a file. Executing a batch Tcl script containing the \$L exportView command requires the GUI to be present.

The view that is exported is based on the specified cell, layer(s), hierarchical depth, and clip size. The size of the output is based on the specified height and width values.

Refer to “[Tcl Lists](#)” on page 22 for information on creating a Tcl list, and to “[Invocation Options](#)” in the *Calibre DESIGNrev Layout Viewer User’s Manual* for information on invoking Calibre DESIGNrev in GUI mode.

Examples

Example 1

This example script exports a view of the topcell to a hierarchical depth of 3. The exported view includes objects on layers 2, 3, 4, and 6 that are within the specified coordinates of the clip. The dimensions of the clip are 13000 nm x 14000 nm and the dimensions of the resulting PNG file are 13000 x 14000 pixels.

```
set L [layout create lab.gds]
set topcell [$L topcell]
$L exportView -format png -filename ./output/$topcell.png \
  -cellname $topcell -layers [list 2 3 4 6] -width 13000 -height 14000 \
  -depth 3 -clip [list 120 350 250 490]
```

\$L extractNet

Performs geometry-based or property-based net extraction.

Usage

```
$L extractNet cellname { {-geom layer x y} | {-prop attr netName} } [-hier 0 | 1]  
[-export layer] [-name netname] [{-rdb | -gds | -oasis} file]
```

Arguments

- ***cellname***
A required argument specifying the name of the top cell on which to run the net extraction.
- { {-geom *layer x y*} | {-prop *attr netName*} }
A required argument choice specifying the type of net extraction to perform. Options are:
 - geom *layer x y* — Performs geometry-based net extraction with connect rules. This argument requires a layer number and x, y coordinates that determine the start location of the geometry to be used for the net extraction.
 - prop *attr netName* — Performs property-based net extraction. The *attr netName* values specify the property name (or GDS attribute number) and property value on which to perform the net extraction.
- -hier 0 | 1
An optional argument specifying the search depth of the net extraction. Options are:
 - 0 — Searches only the top level of the hierarchy.
 - 1 — Searches all levels of the hierarchy. Default.
- -export *layer*
An optional argument used to specify the name of an existing layer in which to export the results of the net extraction. Only full layer numbers (no datatypes) are supported. If no export layer is specified, a layer is automatically chosen by incrementing the base net highlight layer number.
- -name *netname*
An optional argument used to specify a name for the extracted net layer.
- {-rdb | -gds | -oasis} *file*
An optional argument choice used to specify the type of database (ASCII RDB, GDS, or OASIS) in which to write the results of the net extraction.

Return Values

Returns the layer number for the exported net, the number of net segments found, and the layer numbers on which the extracted nets reside. A value of 0 means no connections were found.

Description

The \$L extractNet command provides the ability to perform geometry-based or property-based net extraction.

In GUI mode, you can use connect statements in the layerprops File to define connectivity. In batch mode, you cannot load a layerprops File and therefore must specify the connect rules with a script.

Examples

Example 1

This example runs a hierarchical net extraction from a polygon located at (40000, 687000) in layer 11 inside cell TOP and exports the result to a new layer 100.

```
set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout0
% $L create layer 100
100
% $L extractNet TOP -geom 11 40000 687000 -hier 1 -export 100
Extracting Net ...      Done
34
327 TOP 17.0 19.0 15.0 13.0 11.0 18.0 16.0 14.0 12.0 21.0 20.0
```

Example 2

This example runs a net extraction on the same level of hierarchy from a polygon located at (1400, 6800) in layer 6 and inside cell SUB and exports the result to an RDB file named “net.rdb”.

```
set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$cwb cget -_layout]
layout0
% $L extractNet SUB -geom 6 1400 6800 -hier 0 -rdb net.rdb
Extracting Net ...      Done
Writing file: "net.rdb"
4337 SUB 6.0 7.0 9.0 8.0
```

Example 3

This example uses the \$L connect command to define connectivity between layers 0 and 2.40 by layer 1. The \$L extractNet command then extracts nets from geometries on layer 1 and exports the results to layer 3 and 4. Results are output to an ASCII RDB file and a GDS file.

```
set L [layout create layout0.gds -dt_expand]
$L connect 0 2.40 by 1
$L extractNet TOPCELL -geom 1 500 500 -hier 0 -export 3 \
-rdb ./output/nets.rdb
$L extractNet TOPCELL -geom 1 500 500 -hier 0 -export 4
-gds ./output/layout2.gds
```

Related Topics

[Extracting a Connected Net \[Calibre DESIGNrev Layout Viewer User's Manual\]](#)

[\\$L connect](#)

[\\$L disconnect](#)

[\\$L stopconnect](#)

\$L file

Returns the filename for the current layout.

Usage

\$L file

Arguments

None.

Return Values

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 is returned with the file handle appended to the current working directory.

Examples

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

\$L flatten cell

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

Usage

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

Arguments

- *cellname*
A required argument specifying the name of the cell to flatten.
- -withDelete
An optional argument which causes “newly unreferenced” cell definitions to be deleted from the design due to the flatten operation.

Return Values

None.

Description

Flattens all levels of hierarchy of the specified cell across all instances of the cell. This impacts the cell definition, and hence all references to the cell.

You flatten the hierarchy of a cell to include the cell contents directly into the design, allowing you to edit the geometry without affecting the parent cell, if not flattening the topcell. 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 instance 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.

Examples

Example 1

The following example flattens the cell named “b”. Cell b contains an instance of cell c, which in turn contains an instance of cell d. Flattening cell b results in the references to cells c and d that are contained in cell b to be flattened into b.


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

set lay [layout create "abcd.gds" -dt_expand \
  -preservePaths -preserveTextAttributes]
$lay flatten cell b
$lay gdsout out.gds

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

Example 2

This example flattens the layout:

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

set lay [layout create "abcd.gds" -dt_expand \
  -preservePaths -preserveTextAttributes]
$lay flatten cell topcell
$lay gdsout out.gds

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

Related Topics

[\\$L expand cell](#)

[\\$L flatten ref](#)

\$L flatten ref

Flattens all levels of polygons throughout a cell reference.

Usage

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

Arguments

- **incell**
A required argument specifying the cell containing the reference.
- **refcell**
A required argument specifying the name of the cell being referenced.
- **x y**
A required argument specifying the coordinates of the cell reference to be flattened. By default, coordinates are in database units (dbu).

Coordinates are 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 to be flattened. Valid values include:
 - 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 degrees.
- **mag**
A required argument specifying the magnification to apply to the cell reference. The default is 1, which equates to no magnification. The value of **mag** must be a real number greater than 0.
- *cols rows xspace yspace*
An optional set of arguments that specify the location of a cell reference. Refer to “[\\$L create ref](#)” on page 274 for more information.
- -prop *attr string* [G | U]
An optional argument used to specify a property attribute (PROPATTR) and associated value (PROPVALUE) that must match property data in the specified **refcell** in order for the

reference to be flattened. More than one -prop argument can be specified. Valid options include:

attr — The name of the property attribute (PROPATTR).

string — The value (PROPVVALUE) to assign to the property attribute.

G — Specifies the property is a GDS property. For OASIS databases, this value specifies the property is a standard OASIS property that represents a GDS-style property. This is the default and is the only allowed option for GDS databases.

U — Specifies the property is an OASIS user property. If you write a file containing OASIS user properties to a GDS file, the OASIS properties are discarded.

Return Values

None.

Description

Flattens all levels of any geometry objects throughout the reference. When a cell reference is flattened, the rotation magnification, and mirroring of geometry objects is preserved, but it is not preserved for text objects. This impacts the specified reference, however the cell definition is unaffected.

Examples

Example 1

This example creates a reference of the cell named “dac6_op2” that is in the topcell. The cell reference is located at 238255, 700175 (x, y coordinates) and is not mirrored, rotated, or magnified. The cell reference is then flattened and the results are output to *outfile.oas*.

```
set lay [layout create fullchip.oas]
set top [$lay topcell]
$lay create ref $top dac6_op2 238255 700175 0 0 1
$lay flatten ref top dac6_op2 238255 700175 0 0 1
$lay oasisout ./output/outfile.oas
```

Related Topics

[\\$L expand ref](#)

[\\$L flatten cell](#)

\$L format

Queries the layout file format.

Usage

\$L format

Arguments

None.

Return Values

Returns GDS, OASIS, or OVERLAY file type.

Description

Queries whether the layout file format is GDS, OASIS, or OVERLAY. This command is similar to the “layout peek -format” option.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay format
GDS
```

Related Topics

[layout peek](#)

\$L gdsout

Writes the layout to a GDS-formatted file.

Note



The `-place` and `-texttype` arguments are not supported in HC mode.

Usage

```
$L gdsout fileName [.gz | .Z] [cellName] [-place] [-pcr] [-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 '}'] [-noReport]
```

Arguments

- ***fileName*** [.gz | .Z]

A required argument specifying the name of the file to which the layout is written. If you append a .gz extension to the filename, the file is saved using gzip compression. If you append a .Z to the filename, the file is saved using the “compress” command.

- ***cellName***

An optional argument that writes only the specified cell and its hierarchy to the file. If this argument is not specified, the command writes all cells to the file.

- **-place**

An optional argument that writes only the geometric contents of the specified cell. When no cell is specified, this argument writes the contents of the cell containing the most hierarchy.

Note



This argument is not supported in HC mode.

- **-pcr**

An optional argument that generates a PCR (Peek Cache Repository) file in addition to the exported layout file.

- **-texttype *texttype***

An optional argument specifying a layer datatype to write the output text to. Overrides any datatype mapping that are present.

Note



This argument is not supported in HC mode.

- **-map *L* [*layer* [*datatype*]]**

An optional argument used to specify the name of an original layer *L* that is written to the GDS file. You can optionally specify a different layer (*layer*) and datatype (*datatype*) to map the specified layer (*L*) to in the resulting GDS file. More than one `-map` argument can

be specified. If you do not specify the `-map` keyword, all layers are written. If you specify a datatype for `L`, it should be in the form “*layer.datatype*”.

Note



The layers specified with `-map` are also considered in the determination of whether a cell is empty or not for the purposes of excluding it from being written out. Cells that contain only shapes on layers that are not specified are considered empty and are not written out.

- `-maxPolygonVertices value`

An optional argument used to set a maximum polygon vertex amount that can exist in a single polygon in a layout. If the vertex count for a polygon exceeds the specified *value*, the argument breaks the polygon into smaller polygons that do not exceed the limit. The default is 8190 vertices, with a minimum of 3 and a maximum of 8190.

- `-maxPathVertices value`

An optional argument used to set a maximum path vertex amount that can exist in a single path in a layout. If the vertex count for a path exceeds the specified *value*, this argument breaks the path into smaller chains of paths that do not exceed the limit. The default is 1024 vertices, with a minimum of 2 and a maximum of 1024.

- `-depth depth`

An optional argument and associated depth used to allow writing a layout to a specific cell hierarchy depth. Referenced cells above the specified *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 argument 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 argument used to not write empty cell definitions when writing GDS files. The `noRefs` option suppresses the output of references to empty cells, as well as 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 argument used to write the layout information to a log file. Specify a filename for *logfile* to output the information to a file, or specify “`stdout`” to print the contents to the terminal window.

- `-clips name | -clipsAll | -clipsSplit | -clipsScreen cellname ‘{‘llx lly urx ury’}’`

An optional argument choice that instructs the tool to write one or more clips to a GDS file.

`-clips name` — Writes the clip specified by *name* to a GDS file.


`-clipsAll` — Writes all clips to a GDS file, cutting intersecting geometries.

`-clipsSplit` — Writes all clips to a GDS file, generating a new file for each clip. The format used for each clip filename is: *filename_clipname*.

`-clipsScreen cellname ‘{‘llx lly urx ury’}’` — Writes a clip version of the layout. You can either save a screen view or a region based on the specified coordinates. The coordinates specified are lower left followed by upper right. By default, coordinates are in database units (dbu).

Coordinates are 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.

Note

 Multiple arguments from this set can be specified. If multiple arguments are specified, the last one specified overrides any previously specified arguments. For example, the “-clipsAll” argument overrides the “-clips *name*” argument when the following is specified:

```
$L gdsout file.gds -clips clip1 -clipsAll
```

- `-noReport`

An optional argument used to prevent the transcribing of informational messages, such as “writing file ...”, that are output by the operation. If this argument is not specified, the operation outputs a summary of the results to the Calibre DESIGNrev terminal window.

Return Values

It prints the name of the output GDS file to the transcript (unless `-noReport` is specified).

Description

The `$L gdsout` command writes a layout to the specified GDS-formatted file. This command is not supported when Calibre DESIGNrev is invoked in `-noedit` mode.

Examples

Example 1

This example creates a GDS file named *lab1.gds* and outputs the file to *outfile.gds*.

```
layout create lab1.gds
set lay [lindex [layout all] 0]
$lay gdsout ./output/outfile.gds
Writing GDSII file: "outfile.gds"
```

Example 2

This example outputs a GDS file that contains the area specified by the clip coordinates.

```
set my_layout [layout create lab1.gds]
$my_layout gdsout ./output/my_new_layout.gds -clipsScreen lab1\
{15270 112490 421050 209720}

% set layout [layout create orig.gds]
% layout copy layout0 newlayout topcellname 0 99 x1 y1 x2 y2 2 \
  -preserveHierarchy
% set L newlayout
% $L gdsout outfile2.gds -log outfile2.log
Writing GDSII file: "outfile2.gds"
```

Example 3

This example renames all cells and writes out the new layout:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% set mycells [$lay cells]
% foreach cell $mycells {
    set newname "n20_$cell"
    $lay cellname $cell $newname
}
% $lay gdsout "n20_outfile3.gds"
```

Example 4

These examples write out individual layers:

```
# Write out layer 4
layout0 gdsout mix1.gds -map 4

# Write out layer 4 after mapping it to layer 100
layout0 gdsout mix2.gds -map 4 100

# Write out layer 4 after mapping it to layer 100, datatype 9
layout0 gdsout mix3.gds -map 4 100 9

# Write out layers 1 and 4
layout0 gdsout mix4.gds -map 1 -map 4

# Write out layers 3 & 4, mapping them to 300 and 400.9 respectively
layout0 gdsout mix5.gds -map 3 300 0 -map 4 400 9

# Write out layer 6.12
layout1 gdsout mix6.gds -map 6.12

# Write out layer 6.12 after mapping it to layer 11.9
layout1 gdsout mix7.gds -map 6.12 11 9

# Write out layers 1.5, 3.4 & 5.3,
# mapping them to 105.1, 104.3 & 103.5 respectively
layout2 gdsout mix8.gds -map 1.5 105 1 -map 3.4 104 3 -map 5.3 103 5
```


Related Topics

[\\$L oasisout](#)

[\\$O gdsout](#)

\$L gridsnap

Depending on the arguments, either snaps the geometries and cell references to the specified value of *gridsize* OR checks for violations of a design grid based on *gridsize*.

Usage

\$L gridsnap *gridsize* [-checkonly *outputlayer*] [-c *cellname*] [-reportprogress]

Arguments

- *gridsize*
A required argument specifying the size of the grid to which to snap the geometries and cell references. Specifying this argument only (without -checkonly or -c) modifies the layout by snapping the data on all layers and all placements to the specified value of *gridsize*.
- -checkonly *outputlayer*
An optional argument and value pair that specifies to check for geometries and cell references having coordinates (origins or vertices) that do not fall on the grid points defined by *gridsize*. When specified, any geometry violating the grid is copied to *outputlayer*.
 - When polygons 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 bounding box of an empty cell is (0, 0, 0, 0).The *outputlayer* itself is not checked.
- -c *cellname*
An optional argument that specifies to perform grid checking only on the specified cell rather than the entire layout.
- -reportprogress
An optional argument that displays status messages every few seconds to indicate the progress of the command. The status messages only display if the command takes more than a few seconds. A summary message displays in the Calibre DESIGNrev terminal window when the command completes and overwrites any progress messages.

Return Values

The -checkonly argument returns the number of geometries and cell references that violate the grid.

Description

The `$L gridsnap` command can be used to snap layout data to the specified grid or check for grid violations. This command is limited by the precision of the layout, and specifying a grid size that is smaller than the precision generates a warning. For example:

```
Warning: gridmicrons value <size> is less than the precision of the
layout, value will be rounded.
```

For paths, both the centerline and width are taken into consideration when performing grid-snapping. Any path in which its width extents do not fall on grid points as a result of the grid-snapping operation is converted to a polygon and grid-snapping is applied to the polygon. However, a path with no width is not converted to a polygon prior to grid-snapping. Calibre DESIGNrev generates a warning message when a path is converted to a polygon due to the grid-snapping operation. For example:

```
Warning: 3 paths were converted to polygon(s) in cell TOPCELL due to
gridsnap.
```

The gridsnap operation can also cause path and polygon objects to be deleted from the layout. This occurs when the grid value is large enough to cause all points of an object to collapse down to one grid point. Calibre DESIGNrev generates a warning message when any objects are deleted due to this operation. For example:

```
Warning: 1 object was deleted from cell TOPCELL due to gridsnap.
```

Examples

Example 1

In this example, the `-checkonly` argument checks for geometries and cell references that do not fall on the specified **gridsize**. The results are copied to layer 100 and then output to `mix_gridsnap.gds`.

```
set L [layout create mix.gds -preservePaths -preserveProperties]
$L gridsnap 2 -checkonly 100
$L gdsout ./output/mix_gridsnap.gds
```

Example 2

This example uses the `-checkonly` argument to check for geometries and cell references in cell “xyz” that do not fall on the specified **gridsize**. The results are copied to layer 300 and then output to `layout_gridsnap.oas`.

```
set L [layout create layout.oas -preservePaths -preserveProperties]
$L gridsnap 0.5 -c xyz -checkonly 300 -reportprogress
$L oasisout ./output/layout_gridsnap.oas
```

When the gridsnapping process takes longer than a second, the Calibre DESIGNrev terminal window displays progress report messages similar to the following:

```
gridsnapping cell "xyz" ...
```

Related Topics

[\\$L SNAP](#)

\$L holdupdates

Pauses cell bounding box calculation.

Usage

\$L holdupdates

Arguments

None.

Return Values

None.

Description


Pauses cell bounding box (bbox) calculations. Bbox calculations are 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](#).


Layout viewer bbox calculations are optimized for adding rectangles, polygons, wires, and zero-width paths, even without using the [\\$L holdupdates](#) command. However, when adding shapes to a complex hierarchy of cells, the [\\$L holdupdates](#) command can improve performance. In addition, using the [\\$L holdupdates](#) command when adding text, srefs, and arefs can also improve tool performance.

If [\\$L holdupdates](#) is executed, 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.

Caution

 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.

Note

 [\\$L holdupdates](#) can *only* be called from within a Tcl procedure. The function cannot be called from the command line.

Examples

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 Topics

[\\$L allowupdates](#)

\$L import layout

Imports the specified layout. Options define the manner in which cellname conflicts are resolved.

Note



This command is not supported in HC mode.

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 argument specifying TRUE or FALSE, that 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 tool resolves cell name conflicts. Valid values include:
 - 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 argument used to expand datatypes so that each layer and 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 a previously loaded layout (using a handle); it is only used if the imported layout is a file.

- **-preservePaths**
An optional argument used to preserve path definitions when reading layout files. By default, paths are converted into polygons.

- **-ignoreGdsBoxes**
An optional argument used when creating a layout from a GDS file. Instructs the tool to ignore box records when reading the GDS file.
- **-preserveTextAttributes**
An optional argument instructing the tool to preserve GDS Text Presentation and Strans attribute data when reading data into or writing data from the tool. 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. The default is to preserve these attributes.
- **-preserveProperties**
An optional argument used when merging a layout from a GDS file. Instructs the tool to preserve GDS geometry and reference property data (PROPATTR and 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 argument. If specified, and the exact placements already exist, the command does not append them.

Return Values

None.

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.

Examples

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

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


\$L instancedbout

Writes out instances of the specified cell into ASCII results database.

Usage

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

Arguments

- *fileName*

A required argument specifying the pathname for an ASCII results database (RDB) file in which to write the cell data. This file cannot be appended to by subsequent runs of this command.

- *cellName*

A required argument specifying the name of the cell to search for whose instances are written to the ASCII results database file. You can use the wildcard character (*) to scan multiple cells. The wildcard (*) and backslash (\) are valid cell name characters and when searching for a cell using only the wildcard or backslash, you must escape these characters with a backslash.

- *inCell*

An optional argument specifying the name of a cell other than the topcell from which to write the instance specified by *cellName*.

Return Values

None.

Description

Writes out each instance of the specified cell(s) to an ASCII results database as bounding boxes.

Examples

Example 1

This example outputs all cell instances prefixed with “a12” to the ASCII results database file named *a12.results*.

```
set lay [layout create lab1.gds -preservePaths -preserveProperties]
$lay instancedbout ./output/a12.results a12*
```

Related Topics

[\\$L asciiout](#)

\$L isLayerEmpty

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

Usage

\$L isLayerEmpty *layer_num*

Arguments

- *layer_num*

A required argument specifying the layer number or *layer_number.datatype_number* designation of the layer to check.

Return Values

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

Examples

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

Related Topics

[\\$L exists layer](#)

\$L ismodified

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

Usage

\$L ismodified

Arguments

None.

Return Values

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

Examples

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

\$L isOverlay

Determines if the layout is an overlay.

Usage

\$L isOverlay

Arguments

None.

Return Values

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

Examples

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

Calibre DESIGNrev GUI > Layouts pulldown > overlay0

% set O [$wb cget -_layout]
overlay0

Calibre DESIGNrev GUI > Layouts pulldown > layout0

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

Related Topics

[\\$O layoutHandle](#)

[\\$O layouts](#)

[\\$O overlayCells](#)

[\\$O overlaylayout](#)

[layout overlays](#)

[\\$L isReferenced](#)

[\\$L layerFilters](#)

\$L isReferenced

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

Usage

\$L isReferenced

Arguments

None.

Return Values

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

Examples

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

Calibre DESIGNrev GUI > Layouts pulldown > overlay0

% set O [$wb cget -_layout]
overlay0

Calibre DESIGNrev GUI > Layouts pulldown > layout0

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

Related Topics

[\\$O layoutHandle](#)

[\\$O layouts](#)

[\\$O overlayCells](#)

[\\$O overlayout](#)

[layout overlays](#)

[\\$L isOverlay](#)

[\\$L layerFilters](#)

\$L iterator (poly | wire | text)

Returns a list of the specified type of objects.

Usage

\$L iterator {wire | poly | text} cell list_of_layers range startRange endRange
[-depth startDepth endDepth] [-inCell list_of_cells] [-filterText string]
[-prop propName propValue]] [-includeProperties {0|1}]

Arguments

- **{wire | poly | text}**
A required argument used to specify the type of object that is returned. Valid options are:
 - wire — Returns GDS path objects.
 - poly — Returns polygon and box objects.
 - text — Returns text objects.
- **cell**
A required argument specifying the name of the cell containing the objects to examine.
- **list_of_layers**
A required argument specifying a layer number or layer_number.datatype_number pair for one or more layers containing the objects to count. You can use the Tcl list command to specify multiple layers. The output objects do not include layer information.
- **range startRange endRange**
A required argument 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. Use “end” to specify the last element in the list instead of an integer.
- **-depth startDepth endDepth**
An optional argument specifying the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in the specified **cell**. While traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations.
- **-inCell list_of_cells**
An optional argument specifying a list of cells in which to search for the specified object type. You can specify the name of one cell or use the Tcl list command to specify multiple cells. The value you specify for *list_of_cells* accepts the wildcard (*), which can be used to find zero or more cell occurrences.

This option is only supported when the -depth argument is specified.

- **-filterText *string***

An optional argument used to return Text-type iterators that match the specified *string*. You can specify a single text string or a list of text strings enclosed in braces. The value you specify for *string* accepts the wildcard (*), which can be used to find zero or more text occurrences.

This argument is only valid for Text-type iterators (\$L iterator text) and when -depth is specified. Using this argument with any other iterator types (for example, poly) returns an error.

- **-prop *propName propName* *propValue***

An optional argument specifying a property name and value pair that is attached to shapes or instances. Iterator support of properties is useful to verify net extraction layer results. If this option is used, the iterator only returns objects (geometries or instances) that have the specified *propName* and *propValue* attached. This option is only supported when the -depth option is specified.

- **-includeProperties {0|1}**

An optional argument used to force properties to always or never be included in the result of the iterator. Valid options include:

0 (false) — Force properties to never be included in the Tcl result of the iterator.

1 (true) — Force properties to always be included in the Tcl result of the iterator.

If the -includeProperties option is not specified, properties are included only if the layout is created or loaded with the -preserveProperties option. If you specify “-includeProperties 0” and the layout was created or loaded with -preserveProperties, then no properties are included for each result. If you specify “-includeProperties 1” and the layout was not created or loaded with the -preserveProperties options, empty Tcl lists are included for the properties in each result.

Return Values

Returns a Tcl list containing Tcl objects. Floating point values are returned as Tcl floating point objects. Special characters that require escaping are escaped as necessary. For example, a string object containing a single ‘}’ is 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 — 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}* — An 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 — Each wire returns:

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

where:

- *width* — The path width, in database units (dbu).
 - *pathtype* — The path type values; is 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}* — An 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:

- *text* — The string displayed as text.
- *x y* — The coordinates of the text. Coordinates are in database units (dbu).
- *[presentation [strans magnification angle]]* — The text attribute values, returned only if the text has these attributes associated with it.

Hierarchical Data

- Polygon — Each polygon returns:

$\{[\{properties\}] \{x1\ y1\ x2\ y2\ \dots\ xn\ yn\}\} path\ \{bbox\}\}$

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.
 - $\{x1\ y1\ x2\ y2\ \dots\ xn\ yn\}$ — An ordered list of coordinates of the polygon vertices. Coordinates are in database units (dbu).
 - *path* — The path identifying the cell in which the polygon exists.
 - $\{bbox\}$ — The cell's coordinate space as $\{x\ y\ width\ height\}$.
- Wire — Each wire returns:

$\{\{width\ pathtype\ bgnextn\ endextn\ [\{properties\}] \{x1\ y1\ x2\ y2\ \dots\ xn\ yn\}\} path\ \{bbox\}\}$

where:

- *width* — The path width, in dbu.
 - *pathtype* — The path type values; is 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.
 - $\{x1\ y1\ x2\ y2\ \dots\ xn\ yn\}$ — An ordered list of coordinates of the path vertices. Coordinates are in database units (dbu).
 - *path* — The path identifying the cell in which the wire exists.
 - $\{bbox\}$ — 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:

- *text* — The string displayed as text.
- *x y* — The coordinates of the text. Coordinates are in database units (dbu).

- *[presentation [strans magnification angle]]* — The 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}*.

Description

Outputs a list of objects of the indicated type in the indicated cells and layers. This iterator command returns geometries on the specified layer and cell. Optional arguments let you filter these geometries based on property values, level of hierarchy, and parent cell.

Note



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) 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.

Examples

Example 1

This example copies polygons 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 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 into separate coordinates. The \$L `create polygon` command requires separate values for each coordinate.

Example 2

This example returns all polygons in the cell “nand” that are on layers 1, 2, and 3.

```
% layout0 iterator poly nand [list 1 2 3] range 0 end
```

Example 3

This example returns all text in topcell that are on layer 26 throughout the hierarchy and match the string “vd*”.

```
$L iterator text topcell 26 range 0 end -depth 0 99 -filterText vd*
```

Example 4

This example returns all texts that match the strings “gnd” or “vdd” on layer 26 in the cell named one_bit.

```
$L iterator text one_bit 26 range 0 end -depth 0 99 -filterText [list gnd  
vdd]
```

Example 5

This example returns texts that are on layers 10, 11, and 12, and are in the cells NAND2X1 and NAND3X1.

```
$L iterator text top [list 10 11 12] range 0 end -depth 0 99 \  
-inCell [list NAND2X1 NAND3X1]
```

Related Topics

[\\$L COPY](#)

[layout create \(GDS or OASIS file\)](#)

[\\$L create layer](#)

[\\$L create polygon](#)

[\\$L gdsout](#)

\$L iterator (ref | sref | aref)

Returns a list of the specified type of references.

Usage

\$L iterator [*export layer*] {**ref** | **sref** | **aref**} *cell range startRange endRange*
[-depth *startDepth endDepth* [-inCell *list_of_cells*] [-filterCell *list_of_cells*]
[-prop *propName propValue*]] [-includeProperties {0 | 1}] [-arefToSrefs] [-duplicate]

Arguments

- *export layer*
An optional argument set used to export an iterator result to the specified *layer*.
- {**ref** | **sref** | **aref**}
A required argument used to control which type of references are returned. Valid values include:
 - ref** — Return both single references and arrays.
 - sref** — Return single references.
 - aref** — Return arrays.
- *cell*
A required argument specifying the name of the cell containing the objects to examine.
- *range startRange endRange*
A required argument specifying a starting and ending 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. You can specify “end” for the *endRange* value instead of an integer to specify the last element in the list.
- -depth *startDepth endDepth*
An optional argument specifying the hierarchical output of objects between *startDepth* and *endDepth* with the search beginning in the specified *cell*. When traversing the hierarchy, each instance within an AREF is traversed. This has the effect of hiding AREFs and returning all placement locations.
- -inCell *list_of_cells*
An optional argument specifying a list of cells in which to search for the specified object type. You can specify the name of one cell or use the Tcl list command to specify multiple cells. The value you specify for *list_of_cells* accepts the wildcard (*), which can be used to find zero or more cell occurrences.

This option is only supported when the -depth argument is specified.

- `-filterCell list_of_cells`

An optional argument for SREF- and AREF-type iterators, used to return only objects that reference the specified *list_of_cells*. You can specify the name of one cell or use the Tcl list command to specify multiple cells. The value you specify for *list_of_cells* accepts the wildcard (*), which can be used to find zero or more text occurrences.

This option is only supported when the `-depth` argument is specified.

- `-prop propName propValue`

An optional argument specifying a property name and value pair. The iterator returns only references that match the specified property name and value. This option is only supported when the `-depth` argument is specified.

- `-includeProperties {0 | 1}`

An optional argument used to force properties to always or never be included in the result of the iterator. Valid values include:

0 (false) — Force properties to never be included in the Tcl result of the iterator.

1 (true) — Force properties to always be included in the Tcl result of the iterator.

If the `-includeProperties` option is not specified, properties are included only if the layout is created or loaded with the `-preserveProperties` option. If you specify “`-includeProperties 0`” and the layout was created or loaded with `-preserveProperties`, then no properties are included for each result. If you specify “`-includeProperties 1`” and the layout was not created or loaded with the `-preserveProperties` options, empty Tcl lists are included for the properties in each result.

- `-arefToSrefs`

An optional argument used to return SREF objects representing the internal instances of any AREFs.

- `-duplicate`

An optional argument used to find duplicate references, reporting all duplicated placements.

Return Values

Returns a Tcl list of references that match the specified iterator criteria. Special characters are escaped as necessary. For example, a string object containing a single ‘`’` character is escaped to prevent Tcl from parsing this character as a closing string or command clause. Strings containing special Tcl characters, such as cell names containing ‘`$`’ or property text strings containing blanks, are enclosed in ‘`{ }`’ when necessary.

The data output by this command falls into two different categories: `non_hierarchical` and `hierarchical` data. The format for non-hierarchical data is output when the `-depth` argument is *not* specified and ignores cell contents. The format for hierarchical data is output when `-depth` is specified and returns hierarchy data.

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 single reference information in the following format:

{cell_name x y mirror angle mag [{properties}]}

where:

<i>cell_name</i>	The name of the referenced cell.
<i>x y</i>	The coordinates at which the origin of the cell reference is placed.
<i>mirror</i>	Identifies whether the cell is mirrored around the bottom boundary (lower x-axis when the angle is 0). A value of 0 means the cell is not mirrored and 1 means the cell is mirrored.
<i>angle</i>	The angle of rotation applied to the cell.
<i>mag</i>	The magnification applied to the cell. A value of 1.0 means the cell is not magnified.
<i>{properties}</i>	A list of geometry and reference properties associated with the cell. The layout must be read using the -preserveProperties argument to preserve property data (refer to the layout create commands for information on this argument). An empty list ({}) is returned when the cell has no properties associated with it.

- Aref — Returns array information in the following format:

{cell_name x y mirror angle mag col rows xspace yspace [{properties}]}

where:

<i>cell_name</i>	The name of the cells referenced in the array.
<i>x y</i>	The coordinates at which the origin of the cell reference is placed.
<i>mirror</i>	Identifies whether the cell is mirrored around the bottom boundary (lower x-axis when the angle is 0). A value of 0 means the cell is not mirrored and 1 means the cell is mirrored.
<i>angle</i>	The angle of rotation applied to the cell.
<i>mag</i>	The magnification applied to the cell. A value of 1.0 means the cell array is not magnified.
<i>cols rows</i>	The number of columns and rows in the array.

<i>xspace yspace</i>	The X and Y distance between array elements before magnification.
<i>{properties}</i>	A list of geometry and reference properties associated with the array. The layout must be read using the -preserveProperties argument to preserve property data (refer to the layout create commands for information on this argument). An empty list ({}) is returned when the array has no properties associated with it.

Hierarchical Data

- Sref — Returns single reference information in the following format:

{{cell_name x y mirror angle mag [{properties}]}} path {bbox}}

where:

<i>cell_name</i>	The name of the referenced cell.
<i>x y</i>	The coordinates at which the origin of the cell reference is placed.
<i>mirror</i>	Identifies whether the cell is mirrored around the bottom boundary (lower x-axis when the angle is 0). A value of 0 means the cell is not mirrored and 1 means the cell is mirrored.
<i>angle</i>	The angle of rotation applied to the cell.
<i>mag</i>	The magnification applied to the cell. A value of 1.0 means the cell is not magnified.
<i>{properties}</i>	A list of geometry and reference properties associated with the cell. The layout must be read using the -preserveProperties argument to preserve property data (refer to the layout create commands for information on this argument). An empty list ({}) is returned when the cell has no properties associated with it.
<i>path</i>	The path in which the cell reference resides.
<i>{bbox}</i>	The bounding box information for the placement of the cell. The information is returned as x- and y-coordinates of the lower-left corner of the bounding box, and the width and height.

- Aref — Returns array information in the following format:

{cell_name x y mirror angle mag col rows xspace yspace [{properties}]}} path {bbox}}

where:

<i>cell_name</i>	The name of the cells referenced in the array.
------------------	--

<i>x y</i>	The coordinates at which the origin of the cell reference is placed.
<i>mirror</i>	Identifies whether the cell is mirrored around the bottom boundary (lower x-axis when the angle is 0). A value of 0 means the cell is not mirrored and 1 means the cell is mirrored.
<i>angle</i>	The angle of rotation applied to the cell.
<i>mag</i>	The magnification applied to the cell. A value of 1.0 means the cell array is not magnified.
<i>cols rows</i>	The number of columns and rows in the array.
<i>xspace yspace</i>	The X and Y distance between array elements before magnification.
<i>{properties}</i>	A list of geometry and reference properties associated with the array. The layout must be read using the -preserveProperties argument to preserve property data (refer to the layout create commands for information on this argument). An empty list ({ }) is returned when the array has no properties associated with it.
<i>path</i>	The path in which the cell reference resides.
<i>{bbox}</i>	The bounding box information for the placement of the cell. The information is returned as x- and y-coordinates of the lower-left corner of the bounding box, and the width and height.

Description

Returns a list of the specified type of references. The hierarchical traversal treats instances that are internal to AREFs individually.

Examples

Example 1

This example outputs the number of duplicate references found in each cell in the *top_ic.oas* layout to a *results.txt* file. The duplicate results are also output to an RDB file named *dup2200.rdb*.


```

set mylayout [layout create top_ic.oas -preserveProperties]
set results [open ./output/results.txt w]
$mylayout create layer 2200
set totcnt 0
set cells [layout peek top_ic.oas -cells]
puts $results "The following duplicate references were found:"
foreach cell $cells {
    set cnt [llength [$mylayout iterator export 2200 ref $cell range 0 end \
        -depth 0 9 -duplicate]]
    incr totcnt $cnt
    puts $results "$cnt duplicate references found in $cell"
}
puts $results "Total duplicated $totcnt"
close $results
$mylayout rdbout ./output/dup2200.rdb -cell top -layers 2200

```

Example 2

This example outputs the names of the srefs found in the myram cell to *results.txt*.

```

set mylayout [layout create top_ic.oas -preserveProperties]
set results [open ./output/results.txt w]
set srefs [$mylayout iterator sref [$mylayout topcell] range 0 end \
    -depth 0 99 -filterCell myram* -inCell [list myram]]
puts $results "The following srefs were found in myram:"
    foreach sref $srefs {
        puts $results "$sref"
    }
close $results

```

The *results.txt* contains the following output, with the angle and magnification values shown as floating point values:

```

The following srefs were found in myram:
{myram_bdec 374940 390375 0 0.0 1.0 {}} /top/myram
{374940 390375 324050 8795}
{myram_rdec 374940 407890 0 0.0 1.0 {}} /top/myram
{374940 407890 13490 124045}
{myram_bdec 374940 553935 0 0.0 1.0 {}} /top/myram
{374940 553935 324050 8795}
{myram_rdec 374940 571450 0 0.0 1.0 {}} /top/myram
{374940 571450 13490 124045}

```

Example 3

This example illustrates text string representation, in which braces are used to enclose strings that contain special Tcl characters (for example, `inv[1]` and `nand$1`).

```

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

```

Example 4

This example returns all instances that reference cells containing the string “VIA” and are in the cells “inv” and “nand”.

```
$L iterator ref topcell range 0 end -depth 0 99 -filterCell VIA* \  
-inCell [list inv nand]
```

Example 5

This example returns all instances that reference cells beginning with “M” and that are in cells beginning with “INV” and “BUF”.

```
layout2 iterator ref top range 0 end -depth 0 99 -filterCell M* \  
-inCell [list INV* BUF*]
```

Example 6

The following script (named *cell_size_boundary.tcl*) outputs cell boundary coordinates and cell size in database units (dbu).

```
if {$argc != 1} {  
    puts "Error: wrong # of arguments"  
    #puts "Usage: calibredrv cell_size_boundary.tcl <layout filename>"  
    exit  
}  
set input [lindex $argv 0]  
set L [layout create $input]  
set topcell [$L topcell]  
set cellinfo [$L iterator sref $topcell range 0 end -depth 0 100]  
foreach c $cellinfo {  
    puts "This cell: [lindex $c 0 0] X: [lindex $c 0 1] Y: [lindex $c 0 2]  
        Mirror: [lindex $c 0 3] Angle: [lindex $c 0 4] Mag: [lindex $c 0 5]"  
    puts "Path: [lindex $c 1 0]"  
    puts "Boundary X: [lindex $c 2 0] Y: [lindex $c 2 1]  
        Width: [lindex $c 2 2] Height: [lindex $c 2 3] "  
    puts " "  
    puts "Entire list:    $c"  
}  
}
```

Enter the following command to execute the script:

```
$MGC_HOME/bin/calibredrv cell_size_boundary.tcl <layout_filename>
```

Example output:

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

```

-----
----- LAYOUT FILE OBJECT SUMMARY -----
-----
Type                                Count
-----
LAYER                                10
CELL                                 9
PLACEMENT                           206
ARRAY                                2
RECTANGLE                           723
POLYGON                              52
PATH                                  0
TEXT                                  0
PROPERTY                             0

-----
----- LOAD LAYOUT SUMMARY -----
-----
OASIS FILE READ FROM '<path_to_layout>'

DATABASE PRECISION:  1000

MAXIMUM HIERARCHY DEPTH:  1

LAYOUT FILE SIZE =          12 K  =          0 M
LAYOUT MEMORY    =          357 K  =          0 M
LAYOUT READ TIME = 0 sec REAL.  TOTAL TIME = 0 sec REAL.

This cell: a9500 X: 180500 Y: 449500 Mirror: 0 Angle: 0.0 Mag: 1.0
Path: /lab1
Boundary X: 178500 Y: 447500 Width: 4000 Height: 4000

Entire list:  {a9500 180500 449500 0 0.0 1.0} /lab1 {178500 447500 4000
4000}
This cell: a9500 X: 228500 Y: 449500 Mirror: 0 Angle: 0.0 Mag: 1.0
Path: /lab1
Boundary X: 226500 Y: 447500 Width: 4000 Height: 4000
...

```

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

Returns the number of the specified type of objects.

Usage

\$L iterator count {poly | wire | text} *cell layer* [-depth *startDepth endDepth*]

Arguments

- {poly | wire | text}
A required argument specifying the type of object to be counted. Valid options are:
 - poly — Polygons and boxes.
 - wire — GDS paths.
 - text — Text.
- *cell*
A required argument specifying the name of the cell containing the objects to count.
- *layer*
A required argument specifying the *layer_number* or *layer_number.datatype_number* designation for the layer containing the objects to count.
- -depth *startDepth endDepth*
An optional argument specifying 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.

Return Values

Returns the number of objects of the indicated type in the indicated cells and layers.

Description

Outputs the number of objects of the indicated type in the indicated cells and layers.

Examples

```
% layout0 iterator count wire r2200 poly
2
```

\$L iterator count (ref | sref | aref)

Returns the number of the specified type of references.

Usage

\$L iterator count {ref | sref | aref} cell [-depth *startDepth endDepth*] [-filterCell *cellname*] [-arefToSrefs] [-duplicate]

Arguments

- **{ref | sref | aref}**

A required argument used to control which type of references are returned. Valid values include:

ref — Return both single references and arrays.

sref — Return single references.

aref — Return arrays.

- **cell**

A required argument specifying the name of the cell containing the references to count.

- **-depth *startDepth endDepth***

An optional argument specifying 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.

- **-filterCell *cellName***

An optional argument used to specify the name of a cell for filtering the objects that are counted. The -filterCell option is only valid for SREF and AREF-type iterators, and it is only supported when the -depth option is specified.

The value you specify for *cellName* accepts the wildcard (*), which can be used to find zero or more occurrences of cell references.

- **-arefToSrefs**

An optional argument used to return SREF objects representing the internal instances of any AREFs.

- **-duplicate**

An optional argument used to find duplicate geometries, reporting all duplicated placements.

Return Values

Returns the number of objects of the indicated type in the indicated cells and layers.

Description

Outputs the number of objects of the indicated type in the indicated cells and layers. The hierarchical traversal treats instances internal to AREFs individually.

Examples

Example 1

The example script checks the specified GDS file for empty cells.

```
% cat gdscheck.tcl
# Usage: calibredrv gdscheck.tcl <file.gds>
set gdsfile [lindex $argv 0]
set outf [open ./gdscheck.txt w]
set L [layout create $gdsfile -dt_expand]
set topblock [$L topcell]
set cells [$L cells]
set layers [$L layers]

# Report heading
puts $outf "Input File Path: [$L file]"
puts $outf "\nTopCell: [$L topcell all]"

set extents [$L bbox $topblock]
set lext [split $extents " "]
set llx [expr ( [lindex $lext 0] / 1000 ) ]
set lly [expr ( [lindex $lext 1] / 1000 ) ]
set urx [expr ( [lindex $lext 2] / 1000 ) ]
set ury [expr ( [lindex $lext 3] / 1000 ) ]
puts $outf [format "Extents: %f,%f %f,%f" $llx $lly $urx $ury]

# Report on empty blocks
puts $outf "\nChecking for empty blocks"
foreach block $cells {
    set cnt [$L iterator count ref $block]
    if {$cnt > 0} { continue }
    foreach layer $layers {
        set Pcnt [$L iterator count poly $block $layer]
        set Tcnt [$L iterator count text $block $layer]
        set cnt [expr ($Pcnt + $Tcnt + $cnt)]
    }
    if {$cnt < 1} {puts $outf "$block is empty"}
}
puts $outf "Empty block check complete"

# Report bad block references
puts $outf "\nChecking for bad reference blocks"
foreach block $cells {
    set children [$L children $block]
    foreach child $children {
        set i [lsearch -exact $cells $child]
        if {$i < 0} { puts $outf "$child is not a valid reference in $block!" }
    }
}
puts $outf "Bad reference block check complete"
```

```
# Create output file gdscheck.txt
flush $outf
close $outf
puts "Sent results to gdscheck.txt"
```

To execute the script:

```
calibredrv gdscheck.tcl <filename>.gds
```

The script outputs the following:

```
Input File Path: <filepath>
TopCell: <topcell>
Extents: 0.000000,1.000000 35.000000,34.000000
Checking for empty blocks
Empty block check complete
Checking for bad reference blocks
Bad reference block check complete
```

\$L layerconfigure

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 exist in the layout (default).
- 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.

- **layerNumber**

A required argument specifying the layer number or *layer_number.datatype_number* designation for the layer to be configured.

- *layer_options*

An optional argument specifying the layer property and value pairs to change. Those properties not specified remain unchanged. The layer properties you can configure with this command are:

- fill *fill_color* — The fill color. Values can be any valid tcl color or the pound sign “#” followed by the RGB color representation.
- outline *outline_color* — The outline color. Values can be any valid tcl color or the pound sign “#” followed by the RGB color representation.
- outlinewidth *line_width* — The line width, expressed in pixels.
- stipple *fill_pattern* — The fill pattern can be one of the following: clear, diagonal_1, diagonal_2, wave, brick, circles, speckle (formerly gray50), light_speckle (formerly gray12), solid, or @<pathname> (where pathname is the full path to a .xbm file).
- visible — A Boolean indicating whether the layer is visible (1) or not visible (0).

Return Values

Returns a list of layer properties.

Examples

```
% layout0 layerconfigure 4  
-fill red -outline red -stipple gray12 -visible 1 -outlinewidth 1
```


\$L layerFilters

Defines a layer filter that can be used to filter the layers shown in the Layers Browser.

Usage

```
$L layerFilters [-mdpChipView | -mdpLevelView] {-names [layer] |  
-layers filterName_0...filterName_n | -add filterName [list_of_layers | layer_0...layer_n]}
```

Arguments

- **-mdpLevelView** | **-mdpChipView**
An optional argument used 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*]
An optional argument used to retrieve filter names defined in the layout. Optionally retrieve the filters for a specific layer within the layout.
- **-layers** *filterName_0...filterName_n*
An optional argument used to retrieve a unique set of layers for the specified filter(s). Specifying multiple filters performs an OR of the filter layers.
- **-add** *filterName* [*list_of_layers* | *layer_0...layer_n*]
An optional argument that adds a layer filter named *filterName* to the list of filters for the layout. Options to this argument include:
 - list_of_layer* — Specifies the name of a Tcl list of layers.
 - layer_0...layer_n* — Specifies a list of layers, separated by spaces.

Return Values

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.

Examples

Example 1

This example defines a layer filter that can be used to filter the layers shown in the Layers Browser.

```
% 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
layerFilters -add ma 4
layerFilters -add 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
""
```

Example 2

This example defines a list of layers, then uses the layer list to define the layer filter.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$wb cget -_layout]
layout1
% set layers [list 1 4 17.3 22]
1 4 17.3 22
% $L layerFilters -add jet $layers
% $L layerFilters -names
jet
```

Related Topics

[\\$L isOverlay](#)

[\\$L isReferenced](#)

\$L layernames

Returns a list of layer numbers and layernames in the layout, or assigns a new layer name to the specified layer number.

Usage

Syntax 1

\$L layernames [-regular | -shadow | -all] [*layerNumber* | *layerNumber layerName*]

Syntax 2

\$L layernames -oasis

Arguments

- -regular | -shadow | -all

An optional argument in Syntax 1 specifying the type of layers the command operates on. Allowed values are:

- regular — Operates on only regular layers; those that exist in the layout (default).
- 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.

- *layerNumber* | *layerNumber layerName*

An optional argument choice in Syntax 1 specifying whether the command will return a layername for a specified *layerNumber*, or assign a new *layerName* to a specified *layerNumber*.

layerNumber — Specifies the layer number or *layer_number.datatype_number* designation for the layer whose layername is returned.

layerNumber layerName — Specifies the string to which the layername for *layerNumber* is set.

- **-oasis**

A required argument in Syntax 2 that outputs a list of OASIS layername 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} ...}
```

Return Values


Returns a list of layer number and layername pairs, or none when *layerName* is specified.

Description

This method manages layers within \$L. Its function varies according to the arguments you supply:

- No arguments — Returns a list of layer number and layername pairs, with one pair for every layer in the layout.
- Specify a single layer number — Returns the layername associated with that layer number.
- Specify a single layer number and layername — Sets the layername for that layer number to be *layerName*.
- Specify -oasis — Outputs OASIS layer details.

Note

 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) 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.

Examples

```
% $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
```

\$L layers

Returns a list of all layer numbers in this layout.

Usage

\$L layers [-regular | -shadow | -all] [-cell *cellName*] [-name *layerName*] [-layernum *layerNum*]

Arguments

- -regular | -shadow | -all

An optional argument indicating the type of layers to return in the list.

-regular — Returns only regular layers; those that exist in the layout (default).

-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.

- -cell *cellName*

An optional argument indicating the output is only the layers used in the specified cell, *cellName*.

- -name *layerName*

An optional argument to specify a layername and return the layer number (or layer number and datatype if included).

- -layernum *layerNum*

An optional argument to specify a layer number and returning all layers with the same layer number, including different datatypes.

Return Values

If the -name argument is used, returns the defined layers for the specified layername.

If the -layernum argument is used, returns all layers with the same layer number.

Otherwise, returns list of all the layers in the layout.

Description

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 and datatype pair.

Examples

Example 1

This example obtains a list of regular 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
```

Example 2

Return defined layers for a layername:

```
set lay [layout create]
$lay create layer 123.4
$lay layernames 123.4 dete

$lay layers -name dete
123.4
```

Example 3

Returns all layers with the specified layer number, including different datatypes:

```
$L layers -layernum 3
3 3.1 3.2 3.20 3.200
```

Example 4

This example finds the layers used in each of the cells in a layout:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% set mycells [$lay cells]
lab2 a1500 a1890 a1900 a1960 a2010
% foreach cell $mycells {
    puts "Layers used in cell $cell are: [$L layers -cell $cell]"
}
Layers used in cell lab2 are: 6 2
Layers used in cell a1500 are: 4 2 1
Layers used in cell a1890 are: 8 6 4 2
Layers used in cell a1900 are: 8 2 1
Layers used in cell a1960 are: 8 6 4 2
Layers used in cell a2010 are: 8 6 4
```

Related Topics

[\\$L cells](#)

\$L layoutType

Returns or sets the layout file type.

Usage

\$L layoutType [set {gds | oasis | ascii}]

Arguments

- set {gds | oasis | ascii}

An optional argument specifying the new type (GDS, OASIS, or ASCII) for the layout.

Note



\$L layoutType set is often used when creating a new layout handle.

Return Values

If called with no options, it returns one of the following: gds, oasis, ascii, overlay, mdp.

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.

Examples

```
layout0 layoutType set gds
```

\$L libname

Sets or gets the GDS LIBNAME variable.

Usage

\$L libname [*library_name*]

Arguments

- *library_name*

An optional argument specifying the new value for GDS LIBNAME.

Return Values

Returns the GDS library name if no library name is specified, otherwise there are no returns.

Description

Retrieves the GDS LIBNAME value if no library name is specified; sets the GDS LIBNAME value if one is specified as an argument.

Examples

Example 1

This script opens the layout named *lab1.gds*, outputs the results of the existing GDS library name, then specifies a new value for the GDS library. The library names are written to the file named *lib.txt*.

```
set lay [layout create lab1.gds]
set results [open ./output/lib.txt a]
puts $results [$lay libname]
$lay libname my_layout
puts $results [$lay libname]
close $results
```


\$L loadlayerprops

Loads layer properties from the specified layer property file.

Usage

\$L loadlayerprops *file*

Arguments

- *file*

A required argument specifying the pathname for the layer properties file.

Description

This command loads the following information from the specified layer properties file:

- layer number/layername
- connect statements
- stopconnect statements
- viacell statements

Examples

This example loads the layer properties file named “top_ic.oas.layerprops” into the current layout.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay loadlayerprops top_ic.oas.layerprops
```

Related Topics

[layerprops File Format \[Calibre DESIGNrev Layout Viewer User's Manual\]](#)

\$L MASK_LAYER_INFO

Returns the area of shapes on a layer within a specified cell. Also returns the lower-left coordinates of the layer's extent, along with the width and height of the extent bounding box.

Usage

\$L MASK_LAYER_INFO *cellName* *layer* [*endDepth*]

Arguments

- ***cellName***
A required argument specifying the name of the cell containing the layer to scan. If the specified cell does not exist, the command generates an error.
- ***layer***
A required argument specifying the number of the layer to scan. If the layer contains no objects the area returned is zero. If the layer does not exist, the command generates an error.
- ***endDepth***
An optional value specifying the depth of the cell hierarchy to scan. The default is to scan all levels of the hierarchy.

Return Values

The result is a list in the following format:

area llx lly width height

- *llx* — x coordinate of the lower-left vertex of the layer's extent.
- *lly* — y coordinate of the lower-left vertex of the layer's extent.
- *width* — measure of the extent bounding box along the x-axis.
- *height* — measure of the extent bounding box along the y-axis.

Description

Calculates the area of shapes on a layer within a specified cell. The command flattens all geometries and references of the specified cell and layers (or layer subset, if *endDepth* is specified) to a temporary data structure, and then merges the geometries so that nothing is overlapping. The command then calculates the area based on the current database units.

This command is read-only and does not modify layers. It returns 0 if a layer does not contain objects, or an error if a layer does not exist.

Examples

The following example returns the area of all geometries on layer 1 in the cell named lab2.

```
% set wb [find objects -class cwbWorkBench  
::cwbWorkBench::cwbWorkBench0  
%set lay [$wb cget -_layout]  
layout0  
% layout0 MASK_LAYER_INFO lab2 1  
46259625000.0 20250 19750 415000 368250
```

\$L maxdepth

Returns the maximum depth of the specified cell in the layout.

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.

Return Values

The result is an integer representing the depth.

Description

If a cell name is not specified, the layout viewer returns the highest depth in the design.

Examples

```
% layout0 maxdepth c32d
4
```

\$L modify layer

Changes an existing layer to a new layer or an existing layer and datatype pair to a new layer and datatype pair.

Usage

\$L modify layer *existing_layer new_layer* [*new_layername*]

Arguments

- *existing_layer*
A required argument specifying the existing layer number or layer and datatype pair (for example, *layer.datatype*) in the layout.
- *new_layer*
A required argument specifying the new layer number or layer and datatype pair not present in the layout.
- *new_layername*
An optional argument specifying the layername of the new layer. If not specified the layername is the layer number.

Return Values

None.

Description

Changes an existing layer to a new layer or an existing layer and datatype pair to a new layer and datatype pair. The command deletes the existing layer and creates a new layer. All objects on the existing layer are moved to the new layer. The modified layer change on the layout is propagated to any overlay which this layout is part of.

Examples

Example 1

```
layout0 modify layer 4 404 pol2
```

Example 2

```
layout0 modify layer 1.1 2.2
```

Related Topics

[\\$L create layer](#)

[\\$L delete layer](#)

\$L modify origin

Changes the origin of a specified cell.

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 amount by which to shift the origin of the cell. By default, coordinates are in database units (dbu).
Coordinates are 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 argument that forces the references of the cell (whose origin is being moved) to not be moved within the context of the parent cell.

Return Values

None.

Examples

Example 1

This example moves the origin of the cell named TOPCELL by -6205 dbu in the x-direction and 2505 dbu in the y-direction. The results are output to the file named *TopMoved.oas*.

```
set L [layout create CellsPlaced.oas \  
      -preservePaths -preserveTextAttributes -preserveProperties \  
      -noReport]  
$L modify origin TOPCELL -6205 2505  
$L oasisout ./output/TopMoved.oas
```

Example 2

This example uses the \$L bbox command to return the bounding box of the cell named “Mirror”. The \$L modify origin command then moves the cell “Mirror” by -5005 dbu in the x-direction and 3005 dbu in the y-direction. The \$L bbox command then returns the new origin of the cell and writes the results to the file named *MirrorMoved.oas*.

```
set L [layout create CellsPlaced.oas \  
  -preservePaths -preserveTextAttributes -preserveProperties \  
  -noReport]  
$L bbox Mirror  
-200 -2600 6300 3500  
$L modify origin Mirror -5005 3005  
$L bbox Mirror  
4805 -5605 11305 495  
$L oasisout ./output/MirrorMoved.oas
```

Related Topics

[\\$L cells](#)

\$L modify text

Modifies the layer number, coordinates, text string, presentation (GDS layout only), and text properties of an existing text object.

Usage

\$L modify text *cellname*

layer1 x1 y1 text1 [*presentation1* [*strans1 mag1 angle1*]] {[-prop *attr1 string1* [G|U]]...}
layer2 x2 y2 text2 [*presentation2* [*strans2 mag2 angle2*]] {[-prop *attr2 string2* [G|U]]...}

Arguments

- ***cellname***
A required argument specifying the name of the cell containing the text object you want to modify.
- ***layer1***
A required argument specifying the layer number or layer and datatype pair (for example, *layer.datatype*) containing the text object you want to modify.
- ***x1 y1***
A required argument set specifying the x, y coordinates of the text object you want to modify. By default, coordinates are in database units (dbu).
Coordinates are 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.
- ***text1***
A required argument specifying the text string currently displayed by the text object you want to modify.
- ***presentation1* [*strans1 mag1 angle1*]**
An optional argument set specifying text attribute values of the text object you want to modify. This argument set is only valid for GDS layouts.
This argument set is required when any of the text attributes of the object being modified is set to a non-default value. For example, this happens if the corresponding argument set is used in “\$L create text”. Depending on the non-default values present, you can specify *presentation1* only, or all four arguments in the set.
 - presentation1* — A GDS bit array value that controls text presentation (font and vertical/horizontal).
 - strans1* — A GDS bit array value that controls text transformation (reflection, absolute magnification, and absolute angle).
 - mag1* — A real number that specifies a magnification factor.
 - angle1* — A real number that specifies (in degrees) the angular counterclockwise rotation factor.

- **-prop *attr1 string1* [G|U]**

An optional argument set specifying a property in the form of an attribute-value pair (PROPATTR and PROPVALUE) associated with the text object you want to modify. This argument set can be specified multiple times. Valid options include:

attr1 — An integer identifying the property attribute (PROPATTR).

string1 — An ASCII string that is the value (PROPVALUE) associated with the property attribute (PROPATTR).

G — The property is a GDS property (default). When used with an OASIS database, this option specifies the property is a standard OASIS property that represents a GDS-style property.

U — The property is an OASIS user property. This option cannot be specified for GDS databases. OASIS user properties are discarded when writing to a GDS file.

- ***layer2***

A required argument specifying the layer number or layer and datatype pair (for example, *layer_number.datatype_number*) of the layer that will contain the text object. The specified layer must already exist in the layout. You can specify the same value as ***layer1*** when you are modifying other attributes of the text object and want the text object to remain on the same layer.

- ***x2 y2***

A required argument set specifying the x, y coordinates of the new location for the text object. You can specify the same coordinates as ***x1 y1*** when you are modifying other attributes of the text object and want the coordinates to remain the same. By default, coordinates are in dbu.

Coordinates are 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.

- ***text2***

A required argument specifying the text string for the modified text object. You can specify the same value as ***text1*** when you are modifying other attributes of the text object and want the text string to remain the same.

- ***presentation2* [*strans2 mag2 angle2*]**

An optional argument set specifying text attribute values for the modified text object. This argument set is only valid for GDS layouts. You can specify *presentation2* only, or all four arguments in the set.

If text attributes were specified for the original object, they must also be specified for the modified object. You can specify the same values.

presentation2 — A GDS bit array value that controls text presentation (font and vertical/horizontal direction).

strans2 — A GDS bit array value that controls text transformation (reflection, absolute magnification, and absolute angle).

mag2 — A real number that specifies a magnification factor.

angle2 — A real number that specifies (in degrees) the angular counterclockwise rotation factor.

- `-prop attr2 string2 [G|U]`

An optional argument set specifying a property in the form of an attribute-value pair (PROPATTR and PROPVALUE) associated with the modified text object. This argument set can be specified multiple times. Valid options include:

attr2 — An integer that is the attribute number (PROPATTR).

string2 — An ASCII string that is the value (PROPVALUE) associated with the property attribute (PROPATTR).

G — The property is a GDS property (default). When used with an OASIS database, this option specifies the property is a standard OASIS property that represents a GDS-style property.

U — The property is an OASIS user property. This option cannot be specified for GDS databases. OASIS user properties are discarded when writing to a GDS file.

Return Values

None.

Examples

Example 1

This example creates a text object in the cell named TOP on layer 7 at the (x, y) coordinates 2500, 18000. The attributes for the new text object are AAA (text string), 0 (presentation), 0 (strans), 25 (magnification), and 90 (angle). The \$L modify text command then modifies the text object to move the object to layer 9 and change the coordinates to 3500, 12000, text string to “BBB”, and attributes to 40 (magnification) and 270 (angle).

```
set lay [layout create lab1a.gds]
$lay create text TOP 7 2500 18000 AAA 0 0 25 90
$lay modify text TOP 7 2500 18000 AAA 0 0 25 90 9 3500 12000 BBB 0 0 40 270
```

\$L NOT

Performs a BOOLEAN NOT on the specified layers.

Usage

\$L NOT *InputLayer1 InputLayer2 OutputLayer* [-cell *cell* [-hier {0|1}]]

Arguments

- ***InputLayer1***
A required argument specifying the name of the first layer.
- ***InputLayer2***
A required argument specifying the name of the second layer.
- ***OutputLayer***
A required argument specifying the name of the output layer for the operation. The output layer specified should not also be an input layer.
- **-cell *cell***
An optional argument used to specify the name of a cell in which to perform a Boolean NOT operation. The NOT operation is performed on the current cell unless this argument is specified. An error is returned if the specified *cell* does not exist or is empty.
- **-hier {0|1}**
An optional argument used to enable (1) or disable (0) performing the Boolean NOT operation across all levels of the hierarchy for the specified *cell*. The default (0) disables performing the operation across the hierarchy. When enabled, this argument flattens the portion of the design (based on the specified *cell*) before performing the boolean operation.

Return Values

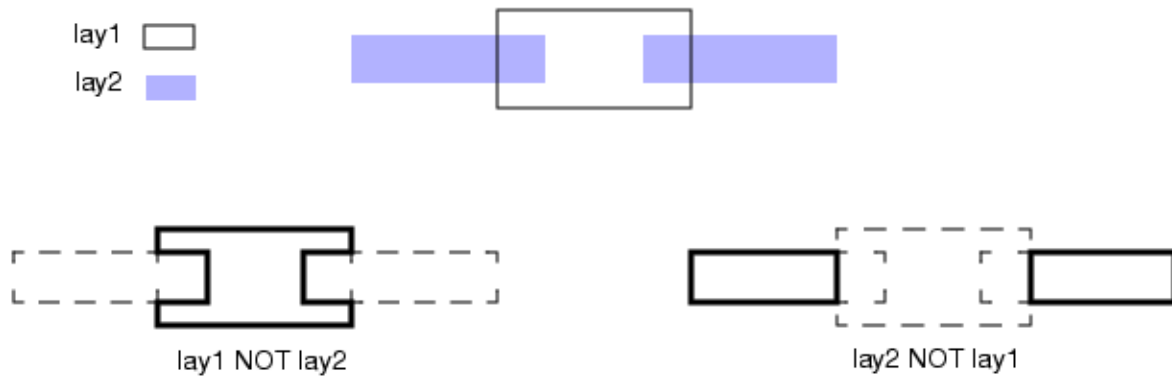
None.

Description

Performs a Boolean NOT on the specified layers and writes the output to the layer ***OutputLayer***. Selects all ***InputLayer1*** polygon areas not common to ***InputLayer2*** polygons. If ***InputLayer2*** is empty, the function generates output equivalent to the polygon data on ***InputLayer1***. The operation can be performed on the current cell or a specified cell.

Figure 5-4 shows two Boolean NOT operations. The left operation selects all lay1 polygon area not common to lay2 polygons. The right operation selects all lay2 polygon area not common to lay1 polygons.

Figure 5-4. Boolean NOT



Examples

Example 1

This example performs a NOT operation on layers 2 and 4, outputs the results to layer 101, and then updates the GUI display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay NOT 2 4 101
% $wb updateDisplay
```

Example 2

This example flattens all levels of the hierarchy in cell a1240, performs a NOT operation on layers 5 and 8, outputs the results to layer 300, and then updates the display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay NOT 5 8 300 -cell a1240 -hier 1
% $wb updateDisplay
```

Related Topics

[\\$L AND](#)


[\\$L OR](#)

[\\$L XOR](#)

\$L oasisout

Writes the layout to an OASIS-formatted file.

Note

 The -place argument is not supported in HC mode.

Usage


\$L oasisout *fileName* [.z|.gz|.Z] [*cellName*] [-place] [-map *L* [*layer* [*datatype*]]]...
[-maxPolygonVertices *value*] [-maxPathVertices *value*] [-depth *depth*] [-fixPolygons]
[-noEmptyCells [noRefs]] {[-pcr] | [-cBlocks] [-compressionLevel { 1 | 9 }]} [-strictMode]}
[-clips *name*] [-clipsAll] [-clipsSplit] [-clipsScreen *cellname* '{' *llx lly urx ury* '}']
[-noRefRepetition] [-noPathRepetition] [-noPolyRepetition] [-noStringRepetition]
[-noRepetition] [-noGdsConvertMsgs] [-noReport] [-log *logfile*]

Arguments

- ***fileName*** [.z|.gz|.Z]

A required argument specifying the name of the file to which the layout is written. You can append a .z or .gz to the filename to save the file using gzip compression, or append a .Z to the filename to save the file using the “compress” command. The gzip and compress commands must be in your path for this compression operation to work.

Note

 Multithreaded readers cannot read files that are compressed using gzip or compress. It is recommended that you use the -cBlocks argument to compress a file that you may want to be read by a multithreaded reader.


- ***cellName***

An optional argument that writes only the specified cell and its hierarchy to the file. If this argument is not specified, the command writes all cells to the file.

- **-place**

An optional argument that writes only the geometric contents of the specified cell. All geometries are flattened to the specified *cellName*, or to the cell containing the most hierarchy when the *cellName* argument is not specified.

Note


 This argument is not supported in HC mode.

- **-map *L* [*layer* [*datatype*]]**

An optional argument used to specify the layer number or layer number and datatype (for example, 33.70) of an original layer *L* to write to the OASIS file. You can optionally specify a different layer (*layer*) and datatype (*datatype*) to map the specified layer (*L*) to in the resulting OASIS file. If you do not specify the -map argument, all layers are written to the

output file. If -map is specified, you must specify it for each layer that you want written to the OASIS file.

Note

 When used with -noEmptyCells, the layers specified by the -map argument are taken into consideration when determining whether or not a cell is empty for the purpose of excluding the cell from being output. Cells that contain only shapes on layers that are not specified by the -map argument are considered empty and are not output.

- **-maxPolygonVertices *value***
An optional argument that sets the maximum number of vertices that are in a single polygon. If a polygon exceeds the specified *value*, this argument breaks the polygon into smaller polygons that do not exceed the limit. The default is 8190 vertices, with a minimum of 3 and a maximum of 8190.
- **-maxPathVertices *value***
An optional argument that sets the maximum number of vertices that can exist in a single path. If a path exceeds the *value*, this argument breaks the path into smaller chains of paths that do not exceed the limit. The default is 1024 vertices, with a minimum of 2 and a maximum of 1024.
- **-depth *depth***
An optional argument used to exclude cell definitions first referenced below the specified *depth*. Referenced cells above *depth* are written with full hierarchy. Referenced cells at *depth* are not written, but references to them are written.
- **-fixPolygons**
An optional argument 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 argument used to exclude the writing of empty cell definitions when writing OASIS files.

The noRefs option excludes the writing of references to empty cells. The noRefs option works recursively, suppressing references to cells that contain references only to empty cells.
- **-pcr**
An optional argument used to create a PCR file in addition to the exported layout file. The format for the PCR filename is *fileName*_oas.pcr.

The -pcr argument cannot be specified with the -strictMode argument. Refer to [“Creating a PCR File for an OASIS Strict Mode Layout”](#) on page 385 for more information.

- **-cBlocks**

An optional argument used to write compression cell blocks (CBLOCKS) at the cell level. This argument is ignored if you output to a .z, .gz, or .Z file.

- **-compressionLevel { 1 | 9 }**

An optional argument that specifies the level of CBLOCK compression used when writing OASIS layout files. This argument has no effect if the -cBlocks argument is not specified.

1 — Uses the highest CBLOCK compression speed.

9 — Uses the highest CBLOCK compression rate.

The value specified with this argument is used instead of the value set in the Preferences dialog box. When this argument is not specified, the oasisout command uses the value set in the dialog box.

When the output format is a gzipped OASIS file (.GZ or .gz), this argument has no effect

- **-strictMode**

An optional argument used to export the layout in OASIS strict mode. Strict mode OASIS layouts have a faster processing time.

Strict mode creates strict mode output for all fields in the table-offsets structure of an OASIS database. With pattern repetition enabled, the TEXTSTRING and PROPSTRING entries in the table-offsets structure have offsets and all references to strings are made by reference-number. Specifying the -noRepetition or -noStringRepetition argument with -strictMode does the following:

- Causes offsets to be set to zero for the TEXTSTRING and PROPSTRING records, which prevents some tools from being able to read the OASIS file.
- Generates the following warning:

Warning: The -strictMode switch requires text repetitions, use of -noRepetition or -noStringRepetition will be ignored.

Note

Layout viewer strict mode arguments are set consistently with Calibre strict mode arguments.

The -strictMode argument cannot be specified with the -pcr argument. Refer to “[Creating a PCR File for an OASIS Strict Mode Layout](#)” on page 385 for more information.

- **-clips *name* | -clipsAll | -clipsSplit | -clipsScreen *cellname* ‘{‘llx lly urx ury’}’**

An optional argument choice that instructs the tool to write one or more clips to a GDS file.

-clips *name* — Write the clip to an OASIS file, cutting intersecting geometries.


-clipsAll — Writes all clips to an OASIS file, cutting intersecting geometries.

-clipsSplit — Writes all clips to an OASIS file, generating a new file for each clip. The format used for each clip filename is: *filename_clipname*.

`-clipsScreen cellname ‘{llx lly urx ury}’` — Saves a clip version for a layout. The argument can either save a screen view or a selected rectangular region. The coordinates specified are lower left followed by upper right. By default, coordinates are in database units (dbu).

Coordinates are 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.

Note

 Multiple arguments from this set can be specified. If multiple arguments are specified, the last one specified overrides any previously specified arguments. For example, the “-clipsAll” argument overrides the “-clips *name*” argument when the following is specified:

```
$L gdsout file.gds -clips clip1 -clipsAll
```

- **-noRefRepetition**
An optional argument that instructs the tool to not add any new reference repetitions (for compression purposes) when writing out the OASIS file. This argument does not remove any existing repetitions from the input database. Arrays with rotated and non-integer scaled cells are created regardless of whether -noRefRepetition is used.
- **-noPolyRepetition**
An optional argument that instructs the tool to not add any new polygon repetitions (for compression purposes) when writing out the OASIS file. This argument does not remove any existing repetitions from the input database.
- **-noPathRepetition**
An optional argument that instructs the tool to not add any new path repetitions (for compression purposes) when writing out the OASIS file. This argument does not remove any existing repetitions from the input database.
- **-noStringRepetition**
An optional argument that instructs the tool to not add any new string repetitions (for compression purposes) when writing out the OASIS file. This argument does not remove any existing repetitions from the input database.
- **-noRepetition**
An optional argument that instructs the tool to not add any new reference, polygon, path, or string repetitions (for compression purposes) when writing out the OASIS file. This argument does not remove any existing repetitions from the input database.

- **-noGdsConvertMsgs**

An optional argument that disables the transcribing of the following message when converting GDS to OASIS:

```
Writing GDS data created with -preserveTextAttributes. The Oasis
format does not support text attribute data.
```

- **-noReport**

An optional argument that prevents the transcribing of informational messages (for example, “writing file ...”) that are output by the operation.

- **-log *logfile***

An optional argument used to write the layout information to a log file. Specify a filename for *logfile* to output the information to a file, or specify “stdout” to output the log information to the terminal window.

Return Values

Prints the name of the output OASIS file to the transcript (unless -noReport is specified).

Description

The \$L oasisout command writes a layout to the specified OASIS-formatted file. This command is not supported when Calibre DESIGNrev is invoked in -noedit mode.

Creating a PCR File for an OASIS Strict Mode Layout

Specifying the -pcr argument with the -strictMode argument is not supported and generates an error. If you require a PCR file for an OASIS strict mode layout, you can create one by incrementally loading the strict mode layout using the -incr argument available with the layout create command. Refer to “[layout create \(GDS or OASIS file\)](#)” on page 161 for more information. You can also use the -handle argument available with the layout peek command to create a PCR file for an OASIS strict mode layout. Refer to “[layout peek](#)” on page 221 for more information.

Examples

Example 1

This example creates a layout from *fullchip.oas* and outputs the layout to *outfile.oas*:

```
set lay [layout create fullchip.oas]
$lay oasisout ./output/outfile.oas
```

Example 2

This example creates a layout from *fullchip.oas* and then outputs the cell named “myram” to *outfile.oas*.

```
set lay [layout create fullchip.oas]
$lay oasisout ./output/outfile.oas myram
```

Example 3

This example creates a layout from *fullchip.oas* and then uses the `-place` argument to output only the geometries in the cell named “myram” to *outfile.oas*.

```
set lay [layout create fullchip.oas]
$lay oasisout ./output/outfile.oas myram -place
```

Example 4

This example creates a layout from *spare_top_wIP.oas* and uses the `-dt_expand` argument to prevent layers with the same layer number (but different datatypes) from being merged. Layers 31 and 33.70 are output to *outfile.oas* and are mapped to layers 310.20 and 330.10, respectively.

```
set lay [layout create spare_top_wIP.oas -dt_expand]
$lay oasisout ./output/outfile.oas -map 31 310 20 -map 33.70 330 10
```

Example 5

In this example, the *fullchip_empty_cells.oas* layout contains both empty cells and references to empty cells. Specifying “`-noEmptyCells noRefs`” excludes the writing of empty cells and any references to empty cells from the resulting *fullchip_norefs.oas* file.

```
layout create fullchip_empty_cells.oas
$lay oasisout ./output/fullchip_norefs.oas -noEmptyCells noRefs
```

Example 6

This example creates an OASIS file named *outfile.oas* and also creates a PCR file named *outfile_oas.pcr*.

```
set lay [layout create fullchip.oas]
$lay oasisout ./output/outfile.oas -pcr
```

Example 7

This example uses the `-cBlocks` argument to write CBLOCKS to the resulting *outfile.oas* file.

```
set lay [layout create fullchip.oas]
$lay oasisout ./output/outfile.oas -cBlocks
```

The [layout peek](#) `-cblockcount` argument returns the number of CBLOCKS in the resulting layout.

```
layout peek outfile.oas -cblockcount
78
```

Example 8

In this example, the [layout peek](#) `-strictmode` argument returns a value of 0 indicating the file *mix.oas* is not in strict mode.

```
layout peek mix.oas -strictmode
0
```

The `$L oasisout -strictMode` argument is then used to output the file *outfile.oas* in OASIS strict mode.

```
set L [layout create mix.oas -dt_expand -preservePaths]
$L oasisout ./output/outfile.oas -strictMode
```

The `layout peek -strictmode` argument is then run on the resulting file *outfile.oas* and returns a value of 1 indicating the file is in strict mode.

```
layout peek outfile.oas -strictmode
1
```

Example 9

This example creates three clips from three cells that are in the layout *fullchip.oas*. The `-clipsSplit` argument is then used to output the clips to separate OASIS files that are named *outfile_Clip_1.oas*, *outfile_Clip_2.oas*, and *outfile_Clip_3.oas*.

```
set lay [layout create fullchip.oas]
$lay create clip dac6_op2
$lay create clip myram
$lay create clip ADC_5bit_sc_5
$lay oasisout ./output/outfile.oas -clipsSplit
```

Related Topics

[\\$L gdsout](#)

[\\$O oasisout](#)

\$L options

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 values can be specified:

- 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 ... layern 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 created from a GDS database.
- 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 created from a GDS database.
- preserveProperties — Reports on whether or not the -preserveProperties option was specified when the layout was created. This option is valid only for layouts created from a GDS database.

Return Values

If no argument is specified, returns a list of options set on the layout. If an argument is specified, returns 0 or 1 (-map returns a string):

- 0 — the option was not used when the layout was created.
- 1 — the option was used when the layout was created.

Examples

```
set lay [layout create fullchip.oas -dt_expand -preservePaths]
$lay options
-dtExpand -preservePaths
```

Related Topics

[layout create Commands](#)

\$L OR

Performs a BOOLEAN OR on the specified layers.

Usage

\$L OR *InputLayer1 InputLayer2 OutputLayer* [-cell *cell* [-hier {0|1}]]

Arguments

- ***InputLayer1***
A required argument specifying the name of the first layer to OR with the specified second layer.
- ***InputLayer2***
A required argument specifying the name of the second layer to OR with the specified first layer.
- ***OutputLayer***
A required argument specifying the name of the output layer for the operation. The output layer specified should not also be an input layer.
- -cell *cell*
An optional argument used to specify the name of a cell in which to perform a Boolean OR operation. The OR operation is performed on the current cell unless this argument is specified. An error is returned if the specified *cell* does not exist or is empty.
- -hier {0|1}
An optional argument used to enable or disable performing the Boolean OR operation across all levels of the hierarchy for the specified *cell*. The default (0) disables performing the operation across the hierarchy. When enabled, this argument flattens the portion of the design (based on the specified *cell*) before performing the boolean operation.

Return Values

None.

Description

Performs a Boolean OR on the specified layers and merges the output to the layer ***OutputLayer***. The operation can be performed on the current cell or a specified cell.

Examples

Example 1

This example performs an OR operation on layers 2 and 4, outputs the results to layer 102, and then updates the GUI display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay OR 2 4 102
% $wb updateDisplay
```

Example 2

This example flattens all levels of the hierarchy in cell a1240, performs an OR operation on layers 5 and 8, outputs the results to layer 300, and then updates the display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay OR 5 8 300 -cell a1240 -hier 1
% $wb updateDisplay
```

Related Topics

[\\$L AND](#)

[\\$L NOT](#)

[\\$L XOR](#)

\$L property

Globally replaces a property in a GDS or OASIS file, or adds a file-level property to an OASIS file.

Note



This command is not supported in HC mode.

Usage

\$L property [{-replace *attr value newValue*} | {-add '{' '{' *attr string* [G|U] '{'
['{' *attr string* [G|U] '{' ...] '}' }' }

Arguments

- -replace *attr value newValue*

An optional argument set that replaces the specified property attribute and value with a new value in a GDSII or OASIS file. You can specify only one attribute value to replace.

attr — Specifies the name of the attribute whose value is to be replaced.

value — Specifies the attribute value to replace.

newValue — Specifies the new value to assign to the attribute.

- -add '{' '{' *attr value* [G|U] '{' ['{' *attr value* [G|U] '{' ...] '}' }

An optional argument set that adds the specified file-level properties (attribute, value, property type) to an OASIS file. File-level properties can only be exported to an OASIS file. You can specify multiple attribute, value, and property type sets when using this argument. Braces are required around the entire set of properties, and each individual set of properties must also be enclosed in braces.

attr — Specifies the name of the property to assign to the file.

value — Specifies a value to assign to the property.

G|U — Specifies the type of property. Use U (default) to indicate an OASIS user-defined property. Use G to indicate a GDS-style property.

Return Values

Returns file properties and property types for an OASIS file if one is specified with no other options.

Description

This command can be used to replace a property attribute value or add file-level properties.

When specified without any arguments, this command returns any file properties for OASIS files. It does not return any property values for GDSII files.

Examples

Example 1

This example uses the `-replace` argument to replace property attribute values in a GDSII file.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$wb cget -_layout]
% $L property -replace type abc xyz
```

Example 2

This example uses the `-add` argument to add the property “1” with a value of “t” to an OASIS file. The `$L property` command when used with no arguments returns the file properties associated with the layout.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set L [$wb cget -_layout]
% $L property -add { { 1 t } }
% $L property
{1 t U}
```

\$L pushmarkers

Pushes the error markers from the top level cell of an ASCII Results Database (RDB) to the cells in which the errors occur. The criteria for pushing markers can be based on an exact match of error markers or a mapping of polygons to error markers.


Usage

```
$L pushmarkers rdb-file [-out rdb-file] [-cell cellName] [-removeDuplicateMarkers]
[-annotateIDs] [-duplicate] [-straddle] [-noEmptyChecks]
[{-match {RuleName | *} matchRDB} | {[ -only] {[ -map {RuleName | *} layer [datatype]]}
...}]
```

Arguments

- ***rdb-file***
A required argument specifying the name of an input RDB file.
- **-out *rdb-file***
An optional argument specifying the output RDB file in which to write the results. If this argument is not specified, the results are written to *output.rdb* by default.
- **-cell *cellName***
An optional argument specifying the name of a top level cell in the input RDB file. This is only used if the name of the top level cell in the input RDB does not match the name of the top level cell in the layout.
- **-removeDuplicateMarkers**
An optional argument used to prevent duplicate markers from occurring at the same location when markers are pushed into subcells. This argument removes duplicate markers when both the marker coordinates and the context cell into which the markers are pushed are identical.

Note

 To ensure that the number of markers in the output file match that of the input file, the input RDB should contain a marker in the top level cell at each location of instances to context cells.

- **-annotateIDs**
An optional argument that adds a user property and ID to each marker in the resulting output file. The property name assigned to each marker is “Orig_ID”, and the value is the original ID of the pushed marker from the input RDB file.
- **-duplicate**
An optional argument that enables each error marker to be duplicated into all levels of the design hierarchy that match the error marker. When this argument is *not* specified, Calibre DESIGNrev pushes results into the lowest child cell containing the error marker.

- -straddle

An optional argument that specifies to push markers that are partially inside an instance to the RDB file. Markers that are adjacent to an instance are ignored.

- -noEmptyChecks

An optional argument that specifies to ignore empty rule check layers in the RDB file. When this argument is specified, Calibre DESIGNrev processes only rule checks in the RDB file that contain results.

- -match {RuleName | *} matchRDB

An optional argument set used to determine if there is an exact match of markers between the input RDB file (*rdb-file*) and the RDB file specified with this argument. If there is an exact match, then the markers in the cell of the input RDB are pushed down into the matching cell context. The matching criteria is based on the marker(s) resulting from a specific rule check or from all rule checks. The -match argument cannot be used with -map. Valid options are:

RuleName | * — A required argument choice specifying to use a specific rule check or all rule checks as the criteria for determining an exact match.

RuleName — Specifies the name of a rule check.

* — Specifies to use all rule checks.

matchRDB — Specifies the name of an RDB file containing markers that are used to compare with the markers in the specified input RDB file for an exact match.

- -only

An optional argument specifying that only the markers from the rules specified by the -map argument are output to the RDB file.

- -map {RuleName | *} layer [datatype]

An optional argument set used to map error markers from one or more specified rules to polygon shapes on a specified layer (and datatype). If polygon shapes are found that interact with error markers, then the markers are pushed from the top level cell down to the cell in which the errors occur. Error markers only get pushed when they interact by touching or intersecting with a polygon shape on the mapped layers. If multiple polygons in different cells interact with a marker, then the markers are moved to the highest level cell that has interacting shapes. Valid options are:

RuleName | * — A required argument choice specifying to use the error markers from a specific rule check or all rule checks to determine interaction with polygon shapes.

RuleName — Specifies the name of a rule check.

* — Specifies to use all rule checks.

layer [datatype] — An argument set specifying the name of a layer, and optionally the datatype, in which to constrain the scan for polygon shapes.

layer — Specifies the name of a layer.

datatype — Specifies the datatype for *layer*.

You can specify the `-map` argument multiple times to include more than one layer to scan for polygon shapes. The `-map` argument cannot be used with `-match`.

Return Values

A summary of the operation is printed at the end of the run. For example:

```
Pushmarkers summary:
-----
Input markers processed   : 13
Pushed markers           : 13
Duplicate markers removed : 0
Total markers written    : 13
```

Description

Pushes error markers from the top level cell in the specified ASCII Results Database (RDB) to the cells in which the errors occur based on the specified matching or mapping criteria. The results are output to a new RDB file, which can then be loaded into Calibre RVE to view the error markers at their new levels.

If `-map` or `-match` is not specified, then all markers that occurred in lower level cells in the hierarchy are pulled to the top level cell in the output.

Examples

Example 1

This example pushes the error markers from the top level cell in the `top.drc.results` database to the cells in which the errors occurred. The `-map` argument maps the error markers from the rule named M5.3 to polygon shapes on layer 3 and then outputs the results to *new.rdb*.

```
set lay [layout create fullchip.oas -preserveProperties -dt_expand]
$lay pushmarkers top.drc.results -out ./output/new.rdb -map M5.3 3 0
```

A summary of the pushed markers is output to the transcript.

```
Pushmarkers summary:
-----
Input markers processed   : 17
Pushed markers           : 6
Duplicate markers removed : 0
Total markers written    : 17
```

Example 2

In this example, the markers in *top.drc.results* that exactly match the markers resulting from all rule checks in *new.rdb* are written to *results.rdb*, and the markers are pushed down into the matching cell context.

```
set lay [layout create fullchip.oas -preserveProperties -dt_expand]
$lay pushmarkers top.drc.results -out ./output/results.rdb \
-match * new.rdb
```

\$L query

Returns information about an object of the specified type in a specified cell, level of hierarchy, and location.

Usage

```
$L query {ref | polygon | wire | edge | vertex | closept | text}  
    cell {startDepth endDepth} {cx cy} [SearchWidth SearchHeight]  
    [-path | -pathlist] [-list] [-inside] [-all]
```

Arguments

- **ref | polygon | wire | edge | vertex | closept | text**

A required argument specifying the object type to query for. Valid options include:

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 the closest point of more than one reference is equidistant from the search coordinates, the query returns a list of these cell references.

polygon — Returns information about the nearest polygon(s), followed by information about a closest point on the polygon. When the closest point of more than one polygon is equidistant from the search coordinates, the query returns a list of these polygons.

wire — Returns information about the path, followed by information about the closest point on the path. When the closest point of more than one wire is equidistant from the search coordinates, the query returns a list of these wires.

edge — Returns information about the nearest edge on the nearest polygon. When the closest point of more than one edge is equidistant from the search coordinates, the query returns a list of these edges.

vertex — Returns information about the nearest vertex on the nearest polygon. When more than one vertex is equidistant from the search coordinates, the query returns a list of these edges.

closept — Returns the nearest point of a visible polygon. When the closest point of more than one polygon is equidistant from the search coordinates, **closept** returns only the first one that it found.

text — Returns the nearest text object. When the closest point of more than one text object is equidistant from the search coordinates, query returns a list containing all these text objects.

- **cell**

A required argument identifying the cell in which to search.

- **startDepth endDepth**

A required argument set that specifies the levels of the hierarchy to search. The query returns objects of the specified type that are on a level between **startDepth** and **endDepth**, inclusive.

- ***cx cy***

A required argument set that specifies the x- and y-coordinates of a point at which to search for the specified object type. When specified with *SearchWidth* and *SearchHeight*, the ***cx*** and ***cy*** arguments define the centerpoint of a rectangular region in which to search for objects of the specified type.

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

- ***SearchWidth SearchHeight***

An optional argument set that defines the width and height of a rectangular region in which to query for objects of the specified type. The centerpoint of the rectangle is defined by the ***cx*** and ***cy*** arguments. The default is 1 micron when these values are not specified.

- **-path | -pathlist**

An optional argument choice that specifies to append the hierarchical path to the returned data. This argument only works for GDS and OASIS layouts, and does not apply to MDP or view-only OASIS data.

-path — Appends the hierarchical path to the returned data.

-pathlist — Appends the hierarchical path as a Tcl list to the returned data.

- **-list**

An optional argument that causes the return list to contain sublists of the matching database objects. This assists with the processing of multiple returned objects. This argument has no effect on the **ref**, **edge**, or **vertex** arguments, as these arguments return sublists of the matching database objects by default.

- **-inside**

An optional argument that checks if the specified coordinates are inside or on the edge of an object.

The -inside option works for all object types, including paths, text, and refs. Zero width paths and text are only returned if the search point is on the center line or origin, and refs are returned if the search coordinates are inside or on the extent (bbox) of the referenced cell.

Note

When -inside is specified, the *SearchHeight* and *SearchWidth* arguments are ignored, and the query returns *all* objects that enclose or touch the search point. The information on which edge or vertex is closest and the actual distance from the search point are also returned with this option.

- **-all**

An optional argument that returns all objects of the specified type that are within the default or specified *SearchHeight* and *SearchWidth* values, in addition to the objects that have the

closest point to the specified **cx** and **cy** values. This argument applies only when **polygon**, **wire**, or **text** is specified.

Return Values

Returns objects found by the query in a Tcl list format. Floating point values are returned as Tcl floating point objects. Special characters are escaped as necessary. For example, a string object containing a single ‘**}**’ is 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 braces when necessary.

The information that is returned varies according to the object type:

- **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* — The lower left corner of the bounding box of the reference.
- *xlen* and *ylen* — The width and height of the bounding box of the reference, respectively.
- *cell_name*, *x*, *y*, *mirror*, *angle*, *mag* — The name of the referenced cell, its x- and y-coordinates, its mirroring (1 indicates that it was mirrored horizontally, 0 indicates no mirroring), orientation in degrees, and magnification 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 and value pairs, returned only if the layout is type GDS and the reference 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* — The layer the polygon is on.

- *{properties}* — A list of property attribute name and value pairs, returned only if the layout is type GDS and the polygon has properties associated with it.
- *{x1 y1 x2 y2 ... xn yn}* — An ordered list of coordinates of the polygon vertices.
- *idx* — The index of the vertex or edge containing the nearest point.
- “v”|“e” — A “v” if the closest point is a vertex, or an “e” if the closest point is on an edge.
- *dist* — The distance between the closest point and the query point.
- *{xpt ypt}* — The 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* — The layer the path is on.
- *width* — The path width.
- *pathtype* — The path type values is 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 and value pairs, returned only if the layout is type GDS and the wire has properties associated with it.
- *{x1 y1 x2 y2 ... xn yn}* — An ordered list of coordinates of the path vertices.
- *idx* — The index of the vertex or edge containing the nearest point.
- “v”|“e” — A “v” if the closest point is a vertex, or an “e” if the closest point is on an edge.
- *dist* — The distance between the closest point and the query point.
- *{xpt ypt}* — The 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 the closest point of more than one polygon is equidistant from the search coordinates, **closept** returns 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]*] — The text attribute values, returned only if the layout is type GDS and the text has these attributes associated with it.

Description

This command outputs all objects and their properties based on the specified cell, level of hierarchy, and location.

Note

 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) 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.

Examples

Example 1

This example illustrates floating point object representation. The example script queries for information about the nearest cell reference(s) for the specified levels of the hierarchy at the x- and y-coordinates of the point at which to search for references:

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
set refList [$lay query ref [$lay topcell] 0 0 368500 17000]
foreach ref $refList {puts $file "{$ref\}`}
close $file
```

The results of the query are output to the file *results.txt*, shown below:

```
{364250 17000 28500 85000 a1220 368500 17000 0 0.0 1.0}
{344250 17000 28500 85000 a1220 368500 17000 1 180.0 1.0}
```

Example 2

This example illustrates text string representation. The example script queries for text in the top cell at the specified x- and y-coordinates:

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
puts $file "[$lay query text [$lay topcell] 0 0 338080 19680]"
puts $file "[$lay query text [$lay topcell] 0 0 356910 19680]"
puts $file "[$lay query text [$lay topcell] 0 0 347695 19680 35000 35000
-all]"
close $file
```

The results of the query are output to the file *results.txt*, shown below:

```
1 338080 19680 {New Text2}
1 356910 19680 {New Text1}
1 356910 19680 {New Text1} 1 338080 19680 {New Text2}
```

Example 3

This example finds and lists all references at a specified point:

```
set lay [layout create example.gds]
set file [open ./output/results.txt w]
set X 2700
set Y 4000
set PREC 1
set start_depth 0
set end_depth 1
set top_cell [$lay topcell]
set refs_at_xy [$lay query ref $top_cell $start_depth $end_depth $X $Y]
puts $file "Following references were found at location [expr $X/$PREC] (x)
[expr $Y/$PREC] (y) under $top_cell:"
foreach ref $refs_at_xy {puts $file "\($ref\)"}
close $file
```

The results of the query are output to the file *results.txt*, shown below:

```
Following references were found at location 2700(x) 4000(y) under TOPCELL:
(204 269 4080 4507 Cell1 167 37 0 0.0 1.0}
(1558 779 4080 4507 Cell1 1521 547 0 0.0 1.0}
(232 1011 4080 4507 Cell1 195 779 0 0.0 1.0}
(532 570 4080 4507 Cell1 495 338 0 0.0 1.0}
```

Example 4

This example queries for references that contain escaped characters in their cell name:

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
set refs [$lay query ref [$lay topcell] 0 1 64500 118000]
foreach ref $refs { puts $file "\{$ref\}" }
close $file
```

The results of the query are output to the file *results.txt*, shown below:

```
{60250 118000 44500 85000 \}_a1240 64500 118000 0 0.0 1.0}
{24250 118000 44500 85000 \}_a1240 64500 118000 1 180.0 1.0}
```

Example 5

This script queries for references inside a rectangle defined by the centerpoint coordinates and the width and height coordinates.

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
set query_result [$lay query ref lab1 0 1 37000 18000 20000 20000]
foreach ref $query_result { puts $file "\{$ref\}" }
close $file
```

The results of the query are output to the *results.txt* file for two references (a1220 and a1310):

```
{36250 17000 28500 85000 a1220 40500 17000 0 0.0 1.0}
{20250 17000 20500 85000 a1310 24500 17000 0 0.0 1.0}
```

Example 6

This script uses the `-inside` argument to return the references that enclose the search coordinates (37000 18000). Coordinates on the edge of a reference are included in the results.

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
set refs ["$lay query ref lab1 0 1 37000 18000 -inside"]
foreach ref $refs { puts $file "\{$ref\}" }
close $file
```

The results of the query are output to the *results.txt* file for two references (a1220 and a1310):

```
{36250 17000 28500 85000 a1220 40500 17000 0 0.0 1.0}
{20250 17000 20500 85000 a1310 24500 17000 0 0.0 1.0}
```

Example 7

This script uses the `-all` argument to return all polygons that overlap the rectangle defined by the centerpoint coordinates and the width and height coordinates.

```
set lay [layout create lab2.gds]
set file [open ./output/results.txt w]
puts $file "Query results without '-all'"
set polyList1 [$lay query polygon lab1 0 1 37000 18000 5000 5000]
for {set i 0} {$i < [llength $polyList1]} {incr i} {
    if {((($i + 1) % 5) == 0)} {
        puts $file "\{[lindex $polyList1 $i]\}"
    } else {
        puts -nonewline $file "\{[lindex $polyList1 $i]\} "
    }
}
puts $file " "
puts $file "Query results using '-all'"
set polyList2 [$lay query polygon lab1 0 1 37000 18000 5000 5000 -all]
for {set i 0} {$i < [llength $polyList2]} {incr i} {
    if {((($i + 1) % 5) == 0)} {
        puts $file "\{[lindex $polyList2 $i]\}"
    } else {
        puts -nonewline $file "\{[lindex $polyList2 $i]\} "
    }
}
close $file
```

The results of the query are output to the *results.txt* file, shown below:

```
Query results without '-all'
{8 24500 17000 40500 17000 40500 29000 24500 29000} {2} {e} {1000.0}
{37000 17000}
{8 24500 17000 40500 17000 40500 29000 28000 29000 28000 45250 25000 45250
25000 29000 24500 29000} {2} {e} {1000.0} {37000 17000}

Query results using '-all'
{1 36250 27000 37750 27000 37750 19750 64500 19750 64500 27000 64750 27000
64750 50000 36250 50000} {4} {v} {1903.943276465977} {37750 19750}
{6 38250 20250 64500 20250 64500 28750 38250 28750} {0} {v}
{2573.9075352467503} {38250 20250}
{8 24500 17000 40500 17000 40500 29000 24500 29000} {2} {e} {1000.0}
{37000 17000}
{8 24500 17000 40500 17000 40500 29000 28000 29000 28000 45250 25000 45250
25000 29000 24500 29000} {2} {e} {1000.0} {37000 17000}
{1 20250 27000 21750 27000 21750 19750 40500 19750 40500 27000 40750 27000
40750 50000 20250 50000} {6} {v} {3913.118960624632} {40500 19750}
{6 22250 20250 40500 20250 40500 28750 22250 28750} {2} {v}
{4160.829244273309} {40500 20250}
```

Example 8

This script queries for a polygon and returns properties associated with that polygon. The `-preserveProperties` argument is required to maintain object properties when loading a layout using `layout create`.

```
set lay [layout create example.gds -preserveProperties]
set file [open ./output/results.txt w]
puts $file "$lay query polygon TOPCELL 0 1 500 500"
close $file
```

The results of the query are output to the *results.txt* file for two references (a1220 and a1310):

```
{0 {{1 test}} 0 0 1000 0 1000 1000 0 1000} 6 v 707.1067811865476 {0 1000}
```

\$L rdbout

Converts all polygons on the specified layers into RDB output.

Usage

\$L rdbout *rdb_filename* **-cell** *cellName* **-layers** *layer_0* [... *layer_n*] [-hier] [-properties]

Arguments

- ***rdb_filename***
A required argument specifying the output Calibre RVE ASCII database to create.
- **-cell *cellName***
A required argument specifying the name of the cell to start scanning the specified layers from.
- **-layers *layer_0* [... *layer_n*]**
A required argument specifying the layers to scan.
- **-hier**
An optional argument specifying to scan in hierarchical mode. The default is flat.
- **-properties**
An optional argument to specify preserving properties when scanning layers.

Return Values

None.

Description

Converts all polygons on the specified layers into RDB output. They can subsequently be viewed with the Calibre RVE graphical debug program.

When layernames are defined, they are output as checknames. If layernames are not available, the command uses the following convention to generate checknames:

LAYER_layer_datatype

Examples

```
layout0 rdbout mylayer1.rdb -cell lab2 -layers 1
```

Related Topics

[\\$L report duplicate ref](#)

\$L readonly

Sets the read-only state for a layout.

Usage

\$L readonly [0 | 1]

Arguments

- 0 | 1

An optional argument used to toggle read-only mode on and off. Valid values include:

0 — Disables read-only mode, allowing the layout to be saved.

1 — Enables read-only mode. When read-only mode is enabled, changes cannot be made to the layout and it cannot be saved.

Return Values

When an argument is not specified, this command returns the current setting (zero or one) of a layout.

Examples

```
% layout0 readonly  
0  
% layout0 readonly 1  
% layout0 readonly  
1
```


\$L refcount

Reports the number of instances of a specified cell or a total count of the instances in the specified parent cell.

Usage

\$L refcount {*cellName* | {-parent *parentCell*} | {-parent *parentCell* *childCell*}}

Arguments

- *cellName* | {-parent *parentCell*} | {-parent *parentCell* *childCell*}

A required argument choice specifying which count of instances the method will return. Valid options include:

cellName — Returns the number of instances of a cell in a layout

-parent *parentCell* — Returns the number of instances contained in a cell.

-parent *parentCell* *childCell* — Returns the number of instances of a child cell found in a parent cell.

Return Values

Returns an integer.

Examples

Example 1

This example returns the number of instances found in “cellb40”.

```
$L refcount -parent cellb40
```

Example 2

This example returns the total number of instances of “celli01” in the design.

```
$L refcount celli01
```

Example 3

This example returns the number of instances of “cellchild” in “topcell”.

```
$L refcount -parent topcell cellchild
```

Related Topics

[layout peek](#)

\$L report duplicate ref

Outputs duplicate cell reference information about a layout.

Usage

\$L report duplicate ref *cell_0* [...*cell_n*] [-useProps] **-out** *fileName*

Arguments

- *cell_0* [...*cell_n*]
A required argument specifying the names of one or more cells to report duplicated placements. The wildcard character ‘*’ is supported, and it finds zero or more cell occurrences. Using ‘*’ in a search for ca*t matches cat, cart, and caught.
- -useProps
An optional argument specifying to consider property values in addition to other parameters of the reference.
- **-out** *fileName*
A required argument specifying the name of the file to contain the ASCII report. Using “stdout” sends the report to the transcript.

Return Values

None.

Examples

```
% layout0 report duplicate ref * -out stdout
Duplicates logfile
References duplicate summary:
-----
Sref duplicates found : 1
Aref duplicates found : 1
```

Related Topics

[\\$L delete duplicate ref](#)

\$L ruler

Imports or exports rulers from a specified cell in the current layout.

Usage

\$L ruler {export | import} -cell *cellname* -path *ruler_file_path*

Arguments

- **export | import**
A required argument that specifies whether to import or export the rulers.
- **-cell *cellname***
A required argument that specifies the name of the cell to use for importing or exporting the rulers.
- **-path *ruler_file_path***
A required argument that specifies the path to the ruler file (*ruler.xml*) for exporting or importing the ruler information.

Return Values

None.

Examples

This example exports rulers from “cellb40” in the current layout to the ruler file named *ruler.xml*.

```
$L ruler export -cell cellb40 -path ruler.xml
```

\$L scale

Scales the layout by the given value.

Usage

\$L scale *scale_factor* [-reportprogress]

Arguments

- *scale_factor*

A required argument specifying a positive floating point number by which the layout is scaled. To scale up a layout, specify a value greater than 1.0. To scale down a layout, specify a value greater than 0 and less than 1.0 (for example, .5).

- -reportprogress

An optional argument that displays status messages every few seconds to indicate the progress of the command. The status messages only display if the command takes more than a few seconds. A summary message displays in the Calibre DESIGNrev terminal window when the command completes and overwrites any progress messages.

Return Values

The number of cells scaled and amount of time to scale the cells are reported when the -reportprogress argument is used.

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 a value.
- Scaling down by a value *scale_factor*, when all data is known to be at coordinates that are multiples of *scale_factor*.

For all other situations, this operation is irreversible.

Examples

Example 1

This example scales layout4 by a multiple of 2 and then updates the display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout4
$lay scale 2
% $wb updateDisplay
```

Example 2

This example scales layout0 by a multiple of .5, and reports the number of cells that were scaled along with the processing time.

```
% layout0 scale .5 -reportprogress
```

When the scaling process takes longer than a second, the Calibre DESIGNrev terminal window displays progress report messages similar to the following:

```
Scaled 7098 cells out of 29666 (23.9% complete)
```

\$L SIZE

Sizes the specified layer.

Usage

\$L SIZE *Lin Lout* **BY** *val*

Note



SIZE and BY must be capitalized.

Arguments

- ***Lin***

A required argument specifying the layer number of the layer to be sized.

- ***Lout***

A required argument specifying 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 remain.

- ***BY val***

A required argument specifying 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.

Return Values

None.

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.

Examples

```
% 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
```

\$L SNAP

Snaps the data in a layer onto a new layer.

Usage

\$L SNAP *Lin Lout*

Arguments

- *Lin*
A required argument specifying the name of the layer to be snapped.
- *Lout*
A required argument specifying the name of the output layer for the operation. The output layer specified should not also be an input layer.

Return Values

None.

Description

Snaps the data in layer *Lin* onto the new layer *Lout*. The SNAP operation is not a hierarchical operation.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% $lay scale [expr 10/9.0]
% $lay SNAP 10 65.4
```

Related Topics

[\\$L gridsnap](#)

\$L srefsFromAref

Returns a list of SREFs converted from an AREF string.

Usage

\$L srefsFromAref *llx lly xlen ylen incell x y mirror angle mag cols rows dx dy* [*{properties}*]
[-noBbox]

Arguments

- *llx lly*
Required argument specifying the lower left corner of the bounding box of the reference. By default, coordinates are in database units (dbu).
- *xlen ylen*
Required argument specifying the width and height of the bounding box of the reference. By default, coordinate lengths are in dbu.
- *incell x y mirror angle mag*
Required argument specifying the name of the referenced cell, coordinates, and orientation in the design. By default, coordinates are in dbu.
- *cols rows dx dy*
Required argument specifying the array information: dimensions (cols rows) and spacing (dx dy). By default, coordinates are in dbu.
- *{properties}*
Optional list of property attribute name and value pairs, specified only if the layout is type GDS and the reference has properties associated with it.
- -noBbox
Optional argument specifying output does not include the transformed SREF bbox data.

Return Values

A list of SREFs. Each SREF is enclosed in braces ({}).

Description

Takes the formatted output from the **\$L query ref** command when run on an AREF. It displays the list of SREF data formatted in the output format from the **\$L query ref** command for an SREF.

Coordinates and coordinate lengths are 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 to remove the enclosing braces ({}) before entering the output as an argument to this command.

Examples

Example 1

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
```

Example 2

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
```

\$L stopconnect

Defines stopping layer(s) for a net connection layer, or returns all possible input or stopping layer combinations.

Usage

\$L stopconnect [*input_layer* [by *stop_layer*]]

Arguments

- *input_layer*
An optional argument that specifies a layername or layer number (with or without a datatype) for the input layer. Wildcards can be used in layer datatype values.
- *by stop_layer*
An optional argument that specifies a layername or layer number (with or without a datatype) for the stop layer. Wildcards can be used in layer datatype values.

Return Values

If this command is used without specifying any arguments, all known input or stopping layer combinations are returned. If an *input_layer* is specified, all stopping layers for the given *input_layer* are returned.

Description

Defines a stop layer for an input layer. All shapes on the specified *input_layer*(s) are cut by shapes on the stopping layer before participating in any net. Shapes on stopping layers can be located anywhere in the hierarchy.

You can specify this command multiple times. This command can also be specified in a layer properties file.

Examples

Example 1

This example defines the M1Stop layer as the stopping layer for M1.

```
$L stopconnect M1 by M1Stop
```

Example 2

This example defines layer 15 as the stop layer for all datatypes of layer 31.

```
$L stopconnect 31.* by 15
```

Example 3

This example defines layer 16 as a stop layer for M1.

```
$L stopconnect M1 by 16
```

Example 4

This example defines a stopping layer in the layer properties file.

```
stopconnect M1 by M1Stop
```

Related Topics

[\\$L connect](#)

[\\$L extractNet](#)

[\\$L disconnect](#)

\$L textout

Displays all text elements in the given cell as a string.

Usage

\$L textout *cellName* [-split]

Arguments

- *cellName*
A required argument specifying the name of the cell containing the text to output.
- -split
An optional argument that formats the output string without braces surrounding the text part:

```
{{layer x y text} ... {layer x y text}}
```

Return Values

Returns all text elements in the given cell as a string formatted as follows:

```
{{layer x y {text}}} ... {layer x y {text}}}
```

Examples

Given a layout with the handle “ar1” and a topcell named “TOP”, the following code displays the text in the topcell and formats the data into individual lines:

```
foreach x [ar1 textout TOP] {puts $x}
```

This example outputs the layer number, X- and Y-coordinates, and text for each object on separate lines:

```
3 45500 150500 {dense_line_end}  
3 52500 150500 {corners}  
3 59500 150500 {corners}  
...  
3 87500 157500 {broken_h}
```

\$L topcell

Returns the topcell name.

Usage

\$L topcell [all]

Arguments

- all

An optional argument used to return a list of the names of all cells not referenced in other cells.

Return Values

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.

Note



Some layout designs, usually those resulting from a merged database, have a structure containing more than one topcell. Unreferenced cells create multiple topcells.

Description

Returns the name of the topcell in \$L. A topcell is a cell that is not referenced within any other cells.

Examples

Example 1

This example finds the topcell in a layout:

```
% layout0 topcell
lab1
```

Example 2

This example finds the topcell and its children:

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout0
% set topc [$lay topcell]
lab1
% $lay children $topc
lab1 a1220 a9500 a1720 a1230 a1310
```

Related Topics

[\\$L children](#)

\$L transcript

Turns recording of edit transcript information on or off.

Usage

\$L transcript {1 | 0}

Arguments

- 0 | 1

A required argument that turns transcript recording off or on, respectively.

0 — Recording of transcript information is turned off.

1 — Recording of transcript information is turned on (default).

Return Values

None.

Description

Turns recording of the edit transcript information on or off. The edit transcript records commands that modify the data in the layout database. The transcript does not include other layout viewer commands, Tel commands, or messages written to the terminal window. The edit transcript includes commands issued by you at the command line as well as those issued by the tool in response to your actions in the GUI.

Use this command with \$L transcript output to write the transcript to a file.

Examples

```
layout0 transcript 0
```

Related Topics

[\\$L transcript output](#)

[\\$L transcript range](#)

[\\$L transcript state](#)

\$L transcript output

Writes the edit transcript to an output file.

Usage

\$L transcript output *fileName*

Arguments

- *fileName*

A required argument specifying the pathname for the file to which the transcript is written.

Return Values

None.

Description

Writes the edit transcript to an output file.

Examples

```
layout0 transcript 1  
layout0 transcript output trans.out
```

Related Topics

[\\$L transcript](#)

[\\$L transcript range](#)

[\\$L transcript state](#)

\$L transcript range

Returns a range of edit transcript lines.

Usage

\$L transcript range *lo hi*

Arguments

- *lo*
A required argument that specifies the first (lowest numbered) line in the transcript to return.
- *hi*
A required argument that specifies the last (highest numbered) line in the transcript to return.

Return Values

A range of transcript lines.

Examples

```
% layout0 delete layer 4  
% layout0 delete layer 11  
% layout0 transcript range 1 10  
layout0 delete layer 4  
layout0 delete layer 11
```

Related Topics

[\\$L transcript](#)

[\\$L transcript output](#)

[\\$L transcript state](#)

\$L transcript state

Returns the edit transcript state (whether recording is on or off).

Usage

\$L transcript state

Arguments

None.

Return Values

1 — Transcript is being recorded.

0 — Transcript is not being recorded.

Examples

```
% 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 Topics

[\\$L transcript](#)

[\\$L transcript output](#)

[\\$L transcript range](#)

\$L units database

Sets or returns the size of database unit in meters.

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. If 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).

Return Values

When no argument is specified, returns the size of the database unit.

Description

This command returns the size of database units in meters. The value of \$L units database is automatically changed if you issue the \$L units microns command.

Examples

```
% layout0 units database  
1e-09
```

Related Topics

[\\$L units database](#)

[\\$L units user](#)

[\\$L units microns](#)

\$L units microns

Sets or returns the number of database units per micron.

Usage

\$L units microns [*units_per_micron*]

Arguments

- *units_per_micron*

An optional argument specifying 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.


Return Values

When no argument is specified, returns the current number of database units per microns.

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 both of the following:

1. Defines the size of the database unit in terms of *database units per micron*.
2. Defines the ratio of user units per database unit to be $1/\text{units_per_micron}$.

It does this by issuing the commands “\$L units user ($1/\text{units_per_micron}$)” and “\$L units database ($10^{-6}/\text{units_per_micron}$)”. Combined, this results in a user unit size of 1 micron.

Examples

Example 1

This example issues 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
```

Example 2

This example issues 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. Also issues the command \$L units database (10⁻⁶/5) or \$L units database 0.0000002. This translates into .20 microns because 10⁻⁶ is one micron, expressed in meters.

Related Topics

[\\$L units database](#)

[\\$L units user](#)

\$L units user

Sets or returns the ratio of user units per database units.

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.

User Units, Precision, and Grids

The SVRF Precision statement 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**

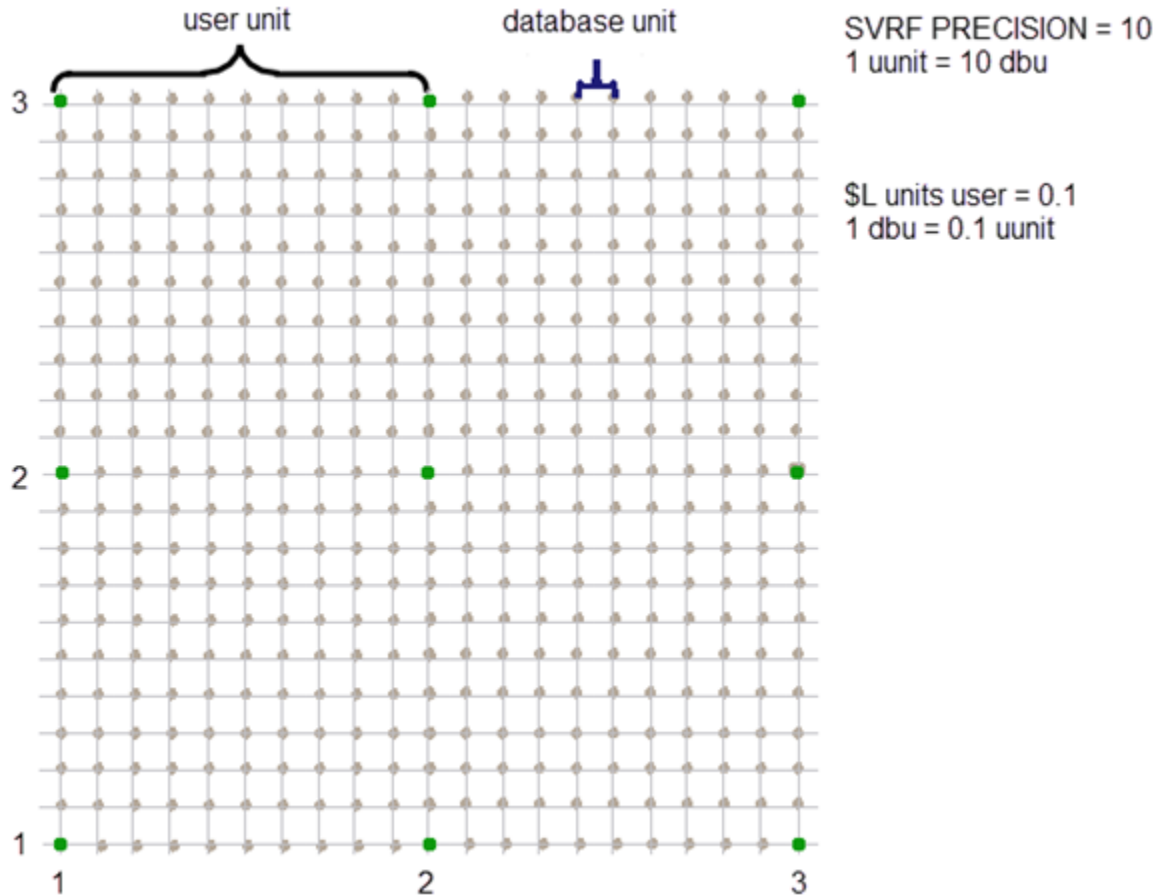
The number of grid points from one user unit to the next.

- **\$L units user**

The distance between grid points expressed in user units.

These results are shown in [Figure 5-5](#).

Figure 5-5. Relating User and Database Units to the Design Grid



Return Values

Returns the ratio of user units per database unit.

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.

Examples

```
% layout0 units user
0.001
```

Related Topics

[\\$L units database](#)

[\\$L units microns](#)

[\\$L units user](#)

\$L update

Updates the layout by recalculating the bounding boxes, topological ordering, and layer tables.

Usage

\$L update

Arguments

None.

Return Values

None.

Examples

```
layout0 update
```

Related Topics

[\\$cwb show](#)

[\\$cwb updateDisplay](#)

\$L viacell

Specifies the named cell is a via cell.

Usage

\$L viacell *cellname*

Arguments

- *cellname*

A required argument specifying the name of the cell to be set as a via cell.

Note



Specifying an asterisk (*) in the cell name matches zero or more cell occurrences, and using it in a search for ca*t finds cat, cart, and cataract.

Return Values

None.

Description

Specifies that the named cell is a via cell. Instances of via cells specified in the layer properties file are 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.

Examples

```
$lay viacell c4400
```

\$L XOR

Performs a BOOLEAN XOR on the specified layers.

Usage

\$L XOR *InputLayer1 InputLayer2 OutputLayer* [-cell *cell* [-hier {0|1}]]

Arguments

- ***InputLayer1***
A required argument specifying the name of the first layer to XOR with the second layer.
- ***InputLayer2***
A required argument specifying the name of the second layer to XOR with the first layer.
- ***OutputLayer***
A required argument specifying the name of the output layer for the operation.
- **-cell *cell***
An optional argument used to specify the name of a cell in which to perform a Boolean XOR operation. The XOR operation is performed on the current cell unless this argument is specified. An error is returned if the specified *cell* does not exist or is empty.
- **-hier {0|1}**
An optional argument used to enable or disable performing the Boolean XOR operation across all levels of the hierarchy for the specified *cell*. Valid values include:
 - 0 — Disables performing the XOR operation across the hierarchy (default).
 - 1 — Enables performing the XOR operation across the hierarchy. When enabled, this argument flattens the portion of the design (based on the specified *cell*) before performing the boolean operation.

Return Values

None.

Description

Performs a Boolean XOR on the specified layers and writes the output to the layer ***OutputLayer***. The operation can be performed on the current cell or a specified cell.

Examples

Example 1

This example performs an XOR on layers 2 and 4, outputs the results to layer 103, and then updates the GUI display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
% $lay XOR 2 4 103
% $wb updateDisplay
```

Example 2

This example flattens all levels of the hierarchy in cell a1240, performs an XOR on layers 5 and 8, outputs the results to layer 300, and then updates the display.

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set lay [$wb cget -_layout]
layout1
%lay XOR 5 8 300 -cell a1240 -hier 1
% $wb updateDisplay
```

Related Topics

[\\$L AND](#)

[\\$L NOT](#)

[\\$L OR](#)

Overlay Commands

Use the overlay commands to manipulate entire overlays, including operations such as creating and merging overlays.

Table 5-10. Overlay Commands Summary

Overlay Command	Description
overlay all	Outputs the handles of all overlay objects existing within the application database.
overlay create	Creates an overlay object from layout handles, and returns an overlay object handle.
overlay delete	Deletes an overlay object.
overlay filemerge	Merges the layout files that are referenced in an overlay and outputs the results to a GDS or OASIS file.

overlay all

Outputs the handles of all overlay objects existing within the application database.

Usage

overlay all

Arguments

None.

Return Values

A list of all overlay object handles. The overlay handles are of the form “overlay N ” where N is an increasing integer starting with zero.

Description

Outputs the handles of all overlay objects existing within the application database.

Examples

```
% overlay all  
olobj1 olobj2 olobj5
```

Related Topics

[layout all](#)

[\\$L format](#)

overlay create

Creates an overlay object from layout handles, and returns an overlay object handle.

Usage

overlay create ‘{’ *layoutHandle* [-cellname {*cellname* | *} [-mirror *mirrorx*] [-angle *angle*] [-mag *mag*] [-align *align*] [-x *dx*] [-y *dy*]] ‘}’ ...

Arguments

- *layoutHandle*

A required argument specifying the layout, by its handle, to include in the overlay being created. When specified without -cellname, the surrounding braces are not needed. For example:

```
% overlay create layout0
```

- -cellname {*cellname* | *}

An optional argument specifying the name of the cell to use. This can be set to either a cell name or the literal ‘*’ as an alias for the topcell.

- -mirror *mirrorx*

An optional argument specifying whether to mirror the layout. Valid options for *mirrorx* include:

- 0 — Do not apply a mirror (default).
- 1 — Mirror across the x axis of the cell.

- -angle *angle*

An optional argument specifying the angle of rotation for the layout. This can be set to 0, 90, 180, or 270 degrees.

- -mag *mag*

An optional argument specifying the magnification to apply to the layout. By default, this is set to 1, which equates to no magnification. The value can be any real number greater than 0.

- -align *align*

An optional argument specifying the alignment to apply to the overlay. Valid values include:

- o none
- o ll (lower left)
- o ul (upper left)
- o ur (upper right)
- o lr (lower right)

- center
- *-x dx*
An optional argument specifying to change the coordinates by the X offset. Coordinates are in database units (dbu).
- *-y dy*
An optional argument specifying to change the coordinates by the Y offset. Coordinates are in dbu.

Return Values

The handle for the new overlay object, such as “overlay0”.

Description

Creates an overlay object from layout handles, and returns an overlay object handle. The command takes one or more layouts, each specified as a Tcl list of a layout handle and zero or more options.

A single layout handle can be specified. Otherwise, the Tcl list must have the layout handle as its first element and each Tcl list starting with ***layoutHandle*** must be enclosed in braces. The remaining elements must be pairs of option names and values, thus the length of the list is always odd. The option pairs can be in any order.

Examples

The following example creates new overlays from layout0 and layout1:

```
% layout create mix.gds
layout0
% layout create compare.gds
layout1

% overlay create layout0
overlay0

% overlay create layout0 layout1
overlay1

% overlay create {layout0 -cellname inv}
overlay2

% overlay create {layout0 -cellname * -mag 1}
overlay3

% overlay create {layout0 -cellname * -mag 1} {layout1 -cellname * -mag 2}
overlay4

% overlay create {layout0 -cellname ws} {layout1 -cellname wb -angle 90}
overlay5

% overlay create {layout0 -cellname ws -angle 0}
{layout1 -cellname wb -mirror 0 -angle 90 -mag 1 -align 0 -x 400 -y 400}
overlay6
```

Related Topics

[layout create Commands](#)

overlay delete

Deletes an overlay object.

Usage

overlay delete *handle*

Arguments

- *handle*

A required argument specifying the handle of the overlay to be deleted.

Return Values

None.

Description

Deletes the overlay specified by *handle*.

Examples

```
% overlay all  
overly2 overly3 overly4 overly5  
  
% overlay delete overly3  
  
% overlay all  
overly2 overly4 overly5
```

Related Topics

[layout delete](#)

overlay filemerge

Merges the layout files that are referenced in an overlay and outputs the results to a GDS or OASIS file.


Usage

overlay filemerge -in *overlay_file* -out *output_file* -createtop *topcell_name* [-integerScaling]

Arguments

- **-in *overlay_file***
A required argument specifying an input overlay filename that references two or more layout files.
- **-out *output_file***
A required argument specifying the name of the layout file in which to output the merged results. When this option is used, the output format of the merged files is determined by the input file format, as follows:
 - Merged GDS input files output GDS format.
 - Merged OASIS input files output OASIS format.
 - Mixed input files (GDS and OASIS) output OASIS format.
- **-createtop *topcell_name***
A required argument that creates a topcell in the output file using the specified ***topcell_name***. The topcells of the layout files referenced in the overlay file are instantiated inside this topcell.

Note

 The ***topcell_name*** must be a unique name that is not used in any of the layout files.

- **-integerScaling**
An optional argument used to set the database precision of the output layout file to the lowest common multiple of all layout files that are referenced by the input overlay file. A positive integer value is used to scale the precisions instead of a fractional number.

The default behavior is to scale the layout files having lower precision values to match that of the layout file with the highest precision value. This can potentially cause problems with some geometry representations. For example, if you merge two layout files whose database precisions are 8 and 20, the layout with the database precision of 8 is scaled by 2.5 so that it also has a database precision of 20.

Using this same example, if -integerScaling is specified, the layout file with a database precision of 8 is scaled by 5 and the layout file with a database precision of 20 is scaled by 2, resulting in a database precision of 40. This method avoids scaling by fractional numbers.

Return Values

This command transcripts the results of the merge operation or returns an error string when an error occurs.

Description

The overlay filemerge command performs a disk-based merge of multiple GDS input files, multiple OASIS input files, or a mixed input format (GDS and OASIS) without loading them into memory. The input overlay must contain two or more layout files.

Examples

Example 1

This example merges the GDS input layouts in the overlay file named *SRAM.ovly* and outputs the results to the OASIS file named *SRAM_ovly_merge.oas*. The topcell in the output OASIS file is named L1.

```
% overlay filemerge -in [list SRAM.ovly] -out SRAM_ovly_merge.oas \  
-createtop L1  
Converting input files :  
Converting GDSII file '<path>/SRAM.gds' to OASIS file '<path>'  
Conversion completed. CPU TIME = 0  REAL TIME = 0  
Merge PASS 1 :  
Creating and sorting index table for file '<path>/SRAM.gds' (1/2)  
Creating and sorting index table for file '<path>/SRAM2.oas' (2/2)  
Merge PASS 2 :  
Processing layername records for file '<path>/SRAM.gds' according to merge  
mode (rename)  
Processing text string records for file '<path>/SRAM.gds' according to  
merge mode (rename)  
Processing property name records for file '<path>/SRAM.gds' according to  
merge mode (rename)  
Processing property string name records for file '<path>/SRAM.gds'  
according to merge mode (rename)  
Processing cells for file '<path>/SRAM.gds' according to merge mode  
(rename)  
Processing layername records for file '<path>/SRAM2.oas' according to  
merge mode (rename)  
Processing text string records for file '<path>/SRAM2.oas' according to  
merge mode (rename)  
Processing property name records for file '<path>/SRAM2.oas' according to  
merge mode (rename)  
Processing property string name records for file '<path>/SRAM2.oas'  
according to merge mode (rename)  
Processing cells for file '<path>/SRAM2.oas' according to merge mode  
(rename)  
Write merge file 'SRAM_ovly_merge.oas'  
Merge completed successfully. CPU TIME = 0  REAL TIME = 0
```

Example 2

This example merges the OASIS input layouts in the overlay file named *fill.ovly* and outputs the results to the OASIS file named *merge.oas*. The *-integerScaling* argument sets the output database precision to 1000, which is the lowest common multiple of the input file precisions.

```
% overlay filemerge -in fill.ovly -out merge.oas -createtop merge_fill
  -integerScaling
Output file precision will be 1000, based on the lowest common multiple of
  the input file precision(s).
Merge PASS 1 :
Creating and sorting index table for file 'spare_top_fill.oas' (1/2)
Creating and sorting index table for file 'spare_top_wIP.oas' (2/2)
Merge PASS 2 :
Processing layername records for file 'spare_top_fill.oas' according to
  merge mode (rename)
Processing text string records for file 'spare_top_fill.oas' according to
  merge mode (rename)
Processing property name records for file 'spare_top_fill.oas' according
  to merge mode (rename)
Processing property string name records for file 'spare_top_fill.oas'
  according to merge mode (rename)
Processing cells for file 'spare_top_fill.oas' according to merge mode
  (rename)
Processing layername records for file 'spare_top_wIP.oas' according to
  merge mode (rename)
Processing text string records for file 'spare_top_wIP.oas' according to
  merge mode (rename)
Processing property name records for file 'spare_top_wIP.oas' according to
  merge mode (rename)
Processing property string name records for file 'spare_top_wIP.oas'
  according to merge mode (rename)
Processing cells for file 'spare_top_wIP.oas' according to merge mode
  (rename)
Write merge file 'merge.oas'
Merge completed successfully. CPU TIME = 3  REAL TIME = 2
```

Related Topics

[layout filemerge](#)

Overlay Objects Methods

Use the overlay object methods to operate on overlays. An overlay object is referenced by its handle, which also acts as a command for operations performed on the overlay. The command reference pages in this section use the \$O variable to refer to the handle. You must assign a value to this variable when executing the command.

Table 5-11. Overlay Object Methods Summary

Overlay Object	Description
\$O gdsout	Saves the overlay to a GDS file.
\$O layoutHandle	Returns the handle for the specified layout ID in the overlay.
\$O layouts	Returns all layouts within the overlay.
\$O oasisout	Saves the overlay to an OASIS file.
\$O overlayCells	Returns all overlaid cells in the overlay.
\$O overlayout	Saves the overlay as an ASCII setup file.

\$O gdsout

Saves the overlay to a GDS file.

Usage

\$O gdsout *filename* [*topcell*]

Arguments

- *filename*

A required argument specifying the pathname for the GDS database file to which the layout is written.

- *topcell*

An optional argument specifying the name of the topcell to use in saving the overlay.

Note



If *topcell* is not specified, the tool uses the overlay cell name as the topcell.

Return Values

None.

Description

For overlays only, saves the overlay to a GDS database file. Only overlays composed of GDS files can be exported as a GDS layout.

When exporting an overlay to a GDS layout file, first save your latest changes for each layout handle of the overlay's layout files, as this command merges layouts using each layout handle's file on disk.

Examples

```
% set cwb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set O [$cwb cget -_layout]
overlay0
% $O gdsout mix
```

Related Topics

[\\$L gdsout](#)

[\\$O oasisout](#)

[\\$O overlayout](#)

\$O layoutHandle

Returns the handle for the specified layout ID in the overlay.

Usage

\$O layoutHandle *overlayLayoutID*

Arguments

- ***overlayLayoutID***
A required argument identifying a layout ID in an overlay for which to return the layout handle.

Return Values

Returns the layout handle for the specified layout ID in the overlay.

Description

Returns the layout handle for a layout ID in the overlay.

Examples

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

% set O [$wb cget -_layout]
overlay0

% $O layouts
L1 L2

% $O layoutHandle L1
layout0
```


\$O layouts

Returns all layouts within the overlay.

Usage

\$O layouts

Arguments

None.

Return Values

Returns the IDs for all layouts within the overlay.

Description

For overlays only, lists all layouts within the overlay.

Examples

This example assumes both a layout and an overlay are open in the GUI. It demonstrates both the proper and improper use of the \$O layouts command.

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

In Calibre DESIGNrev, select overlay0 from the **Layouts** menu.

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

In Calibre DESIGNrev, select layout0 from the **Layouts** menu.

```
% set lay [$wb cget -_layout]
layout0
% $O layouts
L1 L2
% $lay layouts
Invalid layout0 command name "layouts".
```

\$O oasisout

Saves the overlay to an OASIS file.

Usage

\$O oasisout *filename* [*topcell*]

Arguments

- *filename*
A required argument specifying the pathname for the OASIS database file to which the layout is written.
- *topcell*
An optional argument specifying the name of the topcell to use in saving the overlay.

Note



If *topcell* is not specified, the tool uses the overlay cell name as the topcell.

Return Values

None.

Description

For overlays only, saves the overlay to an OASIS database file. Only overlays composed of OASIS files can be exported as an OASIS layout.

When exporting an overlay to an OASIS layout file, first save your latest changes for each layout handle of the overlay's layout files, as this command merges layouts using each layout handle's file on disk.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set O [$wb cget -_layout]
overlay0
% $O oasisout mixoas
```

\$O overlayCells

Returns all overlaid cells in the overlay.

Usage

\$O overlayCells [-layout *overlayLayoutId*]

Arguments

- -layout *overlayLayoutId*

An optional argument specifying the layout ID (*overlayLayoutId*) in the overlay that contains the cells to return.

Return Values

Returns the names of overlaid cells in the overlay.

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.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set O [$wb cget -_layout]
overlay0

% $O layouts
L1 L2
% $O overlayCells
L2-lab1 L1-lab1
% $O overlayCells -layout L1
L1-lab1
```

\$O overlayout

Saves the overlay as an ASCII setup file.

Usage

\$O overlayout [*filename*]

Arguments

- *filename*

An optional argument specifying the name of a ASCII setup data file to save.

Return Values

None.

Description

For overlays only, saves the overlay as an ASCII setup data file.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set O [$wb cget -_layout]
overlay0
% $O overlayout setup.ovly
% ls
setup.ovly
```

Peek Object Methods

Use the peek object methods to operate on a peek object. A peek object is referenced by its handle, which also acts as a command for operations performed on the peek object. The command reference pages in this section use \$P (variable) to refer to the handle. You must assign a value to this variable when executing the command.

For more information on peek objects, refer to [The Peek Cache Repository File](#) in the *Calibre DESIGNrev Layout Viewer User's Manual* and the “[layout peek](#)” on page 221.

Table 5-12. Peek Object Methods Summary

Peek Object Command	Description
\$P delete	Deletes the peek handle. The peek handle is defined by the layout peek -handle command.
\$P file	Returns the filename for a peek handle. The peek handle is defined by the layout peek -handle command.
\$P peek	Returns layout file information for a peek handle. The peek handle is defined by the layout peek -handle command.

\$P delete

Deletes the peek handle. The peek handle is defined by the layout peek -handle command.

Usage

\$P delete [-preservercache]

Arguments

- -preservercache

An optional argument that prevents the cache file from being removed on deletion of the peek handle.

Return Values

None.

Description

Deletes the peek handle defined in a previous layout peek -handle command, to release memory.

Examples

This example creates 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 -format -cellcount]
puts "Cell count: $peekdata(cellcount)"

# Delete the peek handle, preserving cache for subsequent runs.
$peek delete -preservercache
```

Related Topics

[layout peek](#)

[\\$P file](#)

[\\$P peek](#)

\$P file

Returns the filename for a peek handle. The peek handle is defined by the layout peek -handle command.

Usage

\$P file

Arguments

None.

Return Values

The filename for a peek handle.

Description

Returns the filename for a peek handle defined in a previous layout peek -handle command.

Examples

This example creates a peek handle, returns the layout filename, and then deletes it:

```
set layout [lindex lab2.gds]

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

# Return filename for the peek handle.
$peek file

# Delete the peek handle.
$peek delete
```

Related Topics

[layout peek](#)

[\\$P delete](#)

[\\$P peek](#)

\$P peek

Returns layout file information for a peek handle. The peek handle is defined by the layout peek -handle command.

Usage

\$P peek [-bbox *cellname* [-geometryonly]] [-cblock | -cblockcount] [-cellcount] [-cells] [-child *cellname*] [-lastmodtime] [-layers] [-parent *cellname*] [-precision] [-refcount *cellname*] [-topcell] [-topcells] [-indexable] [-format] [-tableoffsets [-*cellname*] [-textstring] [-propname] [-propstring] [-layername] [-xname]] [-strictmode] [-sproperty] [-undefcells] [-units] [-cellhlayers *cellname*] [-celllayers *cellname*]

Arguments

- -bbox *cellname*
An optional argument that queries the specified *cellname* and outputs the bounding box of *cellname* as {*x y width height*}. x and y are the coordinates of the lower left corner of the bounding box and width and height are given in um.
- -geometryonly
An optional argument used to indicate that the -bbox option should ignore text and empty cell references.

Note



This option is only used with the -bbox option. This option is not available with the -cache option.

- -cblock
An optional argument that queries for CBLOCK existence.
- -cblockcount
An optional argument that queries the number of CBLOCKS in this file.
- -cellcount
An optional argument that queries the number of cells in layout.
- -cells
An optional argument that queries all the cell names available in layout.
- -child *cellname*
An optional argument that queries a list of children for the specified *cellname*.
- -lastmodtime
An optional argument that queries for creation date from GDS file. This option only accepts GDS files.

- **-layers**
An optional argument that queries all layers available in layout.
- **-parent *cellname***
An optional argument that queries the parent cell for the specified *cellname*.
- **-precision**
An optional argument that queries the precision, 1/units.
- **-refcount *cellname***
An optional argument that queries a count of instances of references to the specified *cellname*.
- **-topcell**
An optional argument that queries the topcell with the most descendants.
- **-topcells**
An optional argument that queries all topcells available in layout.

Note



Some layout designs, usually those resulting from a merged database, have a structure containing more than one topcell. Unreferenced cells create multiple topcells.

- **-indexable**
An optional argument that queries whether the OASIS file is indexable. OASIS files are indexable if for each cell, the S_CELL_OFFSET property is present in the layout file. If indexable, multiple cells can be read in parallel.
- **-format**
An optional argument that queries whether the file format is GDS or OASIS.
- **-tableoffsets [-cellname] [-textstring] [-propname] [-propstring] [-layername] [-xname]**
An optional argument that queries for all six table-offsets present in the OASIS file, returning a list containing {*tablename table-flag table-offset*} for each of the six table-offsets.

The OASIS table-offsets structure consists of six pairs of unsigned integers. Each pair contains a flag field and a corresponding byte-offset field, as shown in [Table 5-13](#):

Table 5-13. Table Offset Order

Flag	Byte-Offset
cellname flag	cellname offset
textstring flag	textstring offset
propname flag	propname offset

Table 5-13. Table Offset Order (cont.)

Flag	Byte-Offset
propstring flag	propstring offset
layername flag	layername offset
xname flag	xname offset

Note



To use the -tableoffsets option, you should be familiar with OASIS table-offsets. Refer to the OASIS manual for more information concerning the table-offsets structure.

You can also peek data for specific table-offsets using the following options along with the -tableoffsets option:

- -cellname
- -textstring
- -propname
- -propstring
- -layername
- -xname

For example:

```
% $P peek -tableoffsets -cellname -propname {cellname 0 0}  
{propname 0 0}
```

- -strictmode

An optional argument that queries whether the OASIS file is in strict mode (OASIS-STRICT-MODE). The file is in strict mode if the table-offsets flag for all six table offsets is 1. In this case, the -strictmode option returns 1, otherwise the option returns 0.

- -sproperty *property_name*

An optional argument that queries values of OASIS standard properties present in the OASIS layout file. Each standard property name, *property_name*, has a value you can retrieve. The OASIS standard properties are listed in [Table 5-14](#):

Table 5-14. OASIS Properties

Property Name	Value
File-Level Standard Properties	
S_MAX_SIGNED_INTEGER_WIDTH	Bytes to encode a signed integer.
S_MAX_UNSIGNED_INTEGER_WIDTH	Bytes to encode an unsigned integer.

Table 5-14. OASIS Properties (cont.)

Property Name	Value
S_MAX_STRING_LENGTH	Bytes allowed in any string.
S_POLYGON_MAX_VERTICES	Vertices allowed in any polygon.
S_PATH_MAX_VERTICES	Vertices allowed in any path.
S_TOP_CELL	Name(s) of the topcell(s) of a cell hierarchy.
S_BOUNDING_BOX_AVAILABLE	Available bounding boxes. A value of 0 means bounding box properties are not provided. A value of 1 means some bounding box properties are provided. A value of 2 means there is a bounding box property for every CELLNAME record.
Cell-Level Standard Properties	
S_BOUNDING_BOX	Bounding box of the cell. The value consists of the following five fields: flags, lower left X, lower left Y, width, and height.
S_CELL_OFFSET	Byte offset from the beginning of the layout file to the corresponding CELL record. Note: A value of 0 indicates an external cell.

Note

Refer to the OASIS manual for additional information concerning standard properties. The element level property S_GDS_PROPERTY is not supported by the layout viewers.

For the cell-level S_BOUNDING_BOX and S_CELL_OFFSET properties, you can also list the cellname in your query, as these properties are associated with individual cells. If no cellname is specified, the property value for all cells having that property name is returned. Here are the syntax variations available for the -sproperty command:

- o -sproperty *property_name*

This syntax is used to peek any of the OASIS file-level standard properties.

- o -sproperty S_CELL_OFFSET [*cellname1 cellname2 ...*]

This syntax is used to peek the value of the S_CELL_OFFSET property associated with a cell. If no cellname is specified, the tool returns the value of S_CELL_OFFSET for all cells. If the S_CELL_OFFSET property is not present for a given cell, or if the cellname is not present, a warning is generated.

- o -sproperty S_BOUNDING_BOX [*cellname1 cellname2 ...*]

This syntax is used to peek the value of the `S_BOUNDING_BOX` property associated with a cell. If the `S_BOUNDING_BOX` property is not present for the given cell, or if the `cellname` is not present, a warning is generated.

- `-undefcells`
An optional argument that queries cells that are referenced but not defined.
- `-units`
An optional argument that queries the units in microns. Units are displayed if no query option is specified.
- `-celllayers cellname`
An optional argument which returns all layers present in the cell and down its hierarchy.
- `-celllayers cellname`
An optional argument which returns all layers present in the cell.

Return Values

Returns information about the layout file without loading the layout file into memory. The information returned is different for each argument:

- `-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 the layout, including layers that are defined yet contain no shapes. For OASIS, the result for each layer contains the layer number, datatype number, and layername. For GDS, the result for each layer contains the layer number and datatype number.
- `-parent` — Returns the parent cell for the specified *cellname*.
- `-precision` — Returns 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.

- -indexable — Returns 1 if the OASIS file is indexable (for each CELLNAME record, the S_CELL_OFFSET property is present), otherwise the option returns 0.
- -format — Returns GDS or OASIS file type.
- -tableoffsets — Returns list containing {*tablename table-flag table-offset*} for the six table-offsets in the OASIS file.
- -strictmode — Returns 1 if the OASIS file is in strict mode, otherwise the option returns 0. The file is in strict mode if the table-offsets flag for all the 6 offsets tables is 1.
- -sproperty — Does one of the following:
 - -sproperty *property_name* — Returns the OASIS file-level standard property value.
 - -sproperty S_CELL_OFFSET [*cellname1 cellname2 ...*] — Returns the cellname and value. Returns the S_CELL_OFFSET value for all the cells if no cellname is specified
 - -sproperty S_BOUNDING_BOX [*cellname1 cellname2 ...*] — Returns the S_BOUNDING_BOX value for all the cells if no cellname is specified. Otherwise, the result is the cellname and list of 5 bounding box values (flags, lower left X, lower left Y, width, and height).
- -undefcells — Returns cells referenced but not defined.
- -units — Returns the units in microns.
- -cellhlayers — Returns all layers present in the cell and down its hierarchy.
- -celllayers — Returns all layers present in the cell.

If no option is specified, the units are returned.

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.

Examples

This example creates a peek handle, uses the handle to return information about the layout file, and then deletes the peek handle:

```
% set layout [lindex mult.oas]

# Create peek handle.
% set P [layout peek $layout -handle mypeek]

# Get bounding box for topcells.
% array set peekdata [$P peek -format -bbox -topcells]
% set bboxes [$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 mult

% $P peek -child mult
{mult {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 mult}
% $P peek -parent mult
{mult {}}

% $P peek -precision
1000.0

% $P peek -refcount r3500
{r3500 122}
% $P peek -refcount mult
{mult 0}

% $P peek -topcell
mult

% $P peek -topcells
mult add1 add2

% $P peek -indexable
```

```
1

% $P peek -format
OAS

% $P peek -tableoffsets
{cellname 0 0} {textstring 0 0} {propname 0 0} {propstring 0 0}
{layername 0 0} {xname 0 0}

% $P peek -tableoffsets -cellname -propname
{cellname 0 0} {propname 0 0}

% $P peek -strictmode
1

% $P peek -sproperty S_TOP_CELL
TOP

% $P peek -format
OASIS

% $P peek -undefcells
% r1230

% $P peek -units
0.001

% $P peek -cellhlayers mix
{mix {1 0} {2 0} {3 0} {4 0} {5 0} {7 0} {11 0}}
% $P peek -celllayers mix
{mix {7 0} {4 0} {1 0}}

# Delete the peek handle, preserving cache for subsequent runs.
% $P delete
```

Related Topics

[layout peek](#)

[\\$P delete](#)

[\\$P file](#)

[\\$L holdupdates](#)

StringFeature Commands

Use the StringFeature command to create a string feature.

Refer to “[StringFeature Object Methods](#)” on page 467 for information on StringFeatures and the [\\$str addToLayout](#) object method.

Table 5-15. String Feature Commands Summary

String Feature Object	Description
StringFeature	Creates a string feature.

StringFeature

Creates a string feature.

Usage

StringFeature *name x y w h charsize “string” justify*

Arguments

- **name**
A required argument specifying 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**
A required argument specifying 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 \$str addToLayout method. Coordinates are in microns.
- **w h**
A required argument specifying the width and height of the bounding box for the StringFeature. Coordinate lengths are in microns.
- **charsize**
A required argument specifying 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 big to fit the entire string into the bounding box, the tool adjusts the **charsize** down to fit.
- **“string”**
A required argument specifying the text to be represented as polygons. Must be enclosed in quotes, and cannot include newline characters.
- **justify**
A required argument that defines how the string is justified in the bounding box. Valid values include:
 - l — left
 - r — right
 - c — center

Return Values

The handle of the stringfeature.

Examples

The following example creates a StringFeature named abc that contains the string “XYZ string”. The \$str addToLayout command then inserts the string into the TOP cell of layout0.

```
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 have to reload the layout to see the changes.

Related Topics

[\\$str addToLayout](#)

StringFeature Object Methods

Use the StringFeature object methods to operate on StringFeatures. A StringFeature object is referenced by its handle, which also acts as a command for operations performed on the StringFeature. The command reference pages in this section use the \$str variable to refer to the handle. You must assign a value to this variable when executing the command.

The StringFeature object is a set of polygons in the shape of alphanumeric characters, which you can place in a layout using the [\\$str addToLayout](#) method. This differs from the text objects in a layout in that StringFeature objects are geometrical, while text objects are non-geometrical.

Table 5-16. StringFeature Object Methods Summary

StringFeature Object Command	Description
\$str addToLayout	Inserts a StringFeature into an existing layout.

\$str addToLayout

Inserts a StringFeature into an existing layout.

Usage

\$str addToLayout *layout_handle cellname grid layer*

Arguments

- ***layout_handle***
A required argument specifying the handle of the layout into which you want to insert the StringFeature.
- ***cellname***
A required argument specifying the name of the cell into which the StringFeature is inserted.
- ***grid***
A required argument specifying the grid to which the character polygons are snapped, specified in database units (dbu).
- ***layer***
A required argument specifying the layer on which the StringFeature is to be added.

Return Values

None.

Description

Inserts a StringFeature into an existing layout. The variable \$str refers to the handle of an actual StringFeature object to be added to the layout. You use the StringFeature command to create a StringFeature.

Examples

```
$strobj addToLayout layout0 TOP 100 5
```

Related Topics

[StringFeature](#)

cwbLayoutView Object Methods

Use the cwbLayoutView object methods to return information related to the current view of a particular layout. The layout object methods described in this section can be used with all layout viewers.

A layout view object is referenced by its handle, which also acts as a command for operations performed on the layout view object. The command reference pages in this section use the \$V variable to refer to the handle. You must assign a value to this variable when executing the command. Use the command “\$cwb cget -_layoutView” on page 114 to return the handle of the layout object currently visible in the layout viewer main window.

Table 5-17. CwbLayoutView Object Methods Summary

cwbLayoutView Object Command	Description
\$V cell	Returns the name of the currently viewed cell.
\$V databaseToScreenUnits	Translates a pair of X and Y coordinates from database units to screen units.
\$V depth	Returns the current view depth.
\$V exportView	Exports a screen capture of the current view.
\$V micronToScreenUnits	Translates a pair of X and Y coordinates from microns to screen units.
\$V screenToDatabaseUnits	Translates a pair of X and Y coordinates from screen units to database units.

\$V cell

Returns the name of the currently viewed cell.

Usage

\$V cell

Arguments

None.

Return Values

Returns the name of the currently viewed cell.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V cell
a2311
```

\$V databaseToScreenUnits

Translates a pair of X and Y coordinates from database units to screen units.

Usage

\$V databaseToScreenUnits *x y*

Arguments

- *x y*

A required argument specifying the X and Y coordinate values expressed in database units (dbu) you want converted into screen units (px).

Return Values

The X and Y coordinates expressed in screen units.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V databaseToScreenUnits 40489 -39355
300.001055375 600.000661386
```

Related Topics

[\\$V micronToScreenUnits](#)

[\\$V screenToDatabaseUnits](#)

\$V depth

Returns the current view depth.

Usage

\$V depth

Arguments

None.

Return Values

Returns the current view depth.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V depth
4
```


\$V exportView

Exports a screen capture of the current view.

Usage

\$V exportView *imagetype filename*

Arguments

- ***imagetype***

A required argument specifying the image type to output. Valid values include:

jpg — Joint Photographic Experts Group (JPG) is a lossy compression format which supports 8-bit grayscale images and 24-bit color images.

png — Portable Network Graphics (PNG) is both a lossy and lossless compression format which supports 8-bit paletted images and 24-bit color images.

gif — Graphics Interchange Format (GIF) is a lossless compression format limited to an 8-bit palette, or 256 colors.

bmp — Windows bitmap file format (BMP) is an uncompressed format.

xpm — X Pixmap (XPM) is a bitmap format.

ppm — Portable Pixmap file format (PPM) is a bitmap format.

tiff — Tagged Image File Format (TIFF) is both a lossy and lossless compression format which supports 8-bit or 16-bit per color (red, green, blue) images.

- ***filename***

A required argument specifying the filename 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.

Return Values

None.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V exportView gif a1720.gif
% ls
a1720.gif
...
```

\$V micronToScreenUnits

Translates a pair of X and Y coordinates from microns to screen units.

Usage

\$V micronToScreenUnits *x y*

Arguments

- *x y*

A required argument specifying the *x* and *y* coordinate values in microns converted into screen units (px).

Return Values

The X and Y coordinates expressed in screen units.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V micronToScreenUnits 300 600
1037.68336634 -1217.42085032
```

Related Topics

[\\$V databaseToScreenUnits](#)

[\\$V screenToDatabaseUnits](#)

\$V screenToDatabaseUnits

Translates a pair of X and Y coordinates from screen units to database units.

Usage

\$V screenToDatabaseUnits *x y*

Arguments

- *x y*

A required argument specifying the *x* and *y* coordinate values expressed in screen units (px) you want converted into database units (dbu).

Return Values

The X and Y coordinates expressed in database units.

Examples

```
% set wb [find objects -class cwbWorkBench]
::cwbWorkBench::cwbWorkBench0
% set V [$wb cget -_layoutView]
::cwbLayoutView0
% $V screenToDatabaseUnits 300 600
40489 -39355
```

Related Topics

[\\$V databaseToScreenUnits](#)

[\\$V micronToScreenUnits](#)

Macros Commands

Macros are application extensions you write to add new functionality to your application. In order to use macros, you must first add them to the Macros menu.

For a complete discussion of creating macros, refer to “[Writing Macros](#)”.

Table 5-18. Macros Commands Summary

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

Macros create

Creates and registers 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 name of the Tcl procedure that performs the macro processing you require.

For example, this basic ***commandProc*** procedure displays a message:

```
proc my_cmd_procedure { } {  
    tk_messageBox -message "Called my_cmd_procedure." -type ok  
    ...  
}
```

- ***guiPlugin***

A required argument specifying the Tcl procedure that interfaces to the Calibre layout viewer GUI. The example GUI plug-in procedure requires two arguments: the handle for the layout viewer tool and the handle for the window from which the macro is invoked.

For example, this ***guiPlugin*** procedure invokes the ***commandProc*** procedure from the GUI plug-in:

```
proc my_plugin_procedure { wbHandle window } {  
    my_cmd_procedure  
}
```

Return Values

None.

Description


Creates a macro and registers it with a Calibre layout viewer. The new macro adds functionality to the layout viewer. You write the required macro procedures using the Tcl programming language. Once registered, the macro name can be placed on the **Macros** menu as a menu item. A common use for macros is to provide access to Calibre batch tools.

This command requires two existing Tcl procedures, one containing the commands you want the macro to perform and the other to interface to the Calibre layout viewer GUI.

In this example, the **Macros create** command registers the “my_cmd_procedure” Tcl procedure as a macro:

```
if {[info exists tk_version]} {  
    Macros create "my macro" my_cmd_procedure my_plugin_procedure  
}
```

Tip


 The **Macros** Tcl command is only available when running the GUI and not in shell mode, therefore it is recommended that you wrap macro definitions in an “if {[info exists tk_version]} {...}” block as shown above to avoid errors when the *wbinit.tcl* file is sourced.

In order to add a macro to the Calibre layout viewer **Macros** menu, use the **source** command to source the macro from your *\$HOME/wbinit.tcl* file, and then invoke the layout viewer.

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

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

Note

 For more information on writing macros, see “[About Writing Macros](#)” on page 72.

Examples

Example 1

Assuming you have a Tcl procedure which you want made into a macro, you have only to add a second brief procedure and a call to the **Macros create** batch command to turn the original procedure into a macro.

In the following example, a Tcl procedure has been expanded into a macro:

```
proc original_tcl_script { } {  
    tk_messageBox -message "Testing..." -type ok  
    ...  
}  
proc new_macro_plug_in { wbHandle window} {  
    original_tcl_script  
}  
if {[info exists tk_version]} {  
    Macros create "my menu title" original_tcl_script new_macro_plug_in  
}
```

To add the macro to the layout viewer **Macros** menu, source the macro from your *wbinit.tcl* file, and then invoke the Calibre layout viewer as always:

```
echo "source macro_file.tcl" >> ~/.calibrewb_workspace/wbinit.tcl  
calibredrv
```

Related Topics

[Macros delete](#)

[Macros menu](#)

[Macros show](#)

Macros delete

Removes (un-registers) the specified macro.

Usage

Macros delete *macroName*

Arguments

- *macroName*

A required argument specifying the name of the macro to be deleted.

Return Values

None.

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.

Examples

```
Macros delete menu_title
```

Related Topics

[Macros create](#)

[Macros menu](#)

[Macros show](#)

Macros menu

Re-creates the Macros menu with all registered macros.

Usage

Macros menu

Arguments

None.

Return Values

None.

Description

Re-creates the **Macros** menu with all your currently defined macros added.

Examples

Macros Menu

Related Topics

[Macros create](#)

[Macros delete](#)

[Macros show](#)

Macros show

Returns information about macros currently registered with the tool.

Usage

Macros show [*macroName*]

Arguments

- *macroName*

An optional argument specifying the name of the macro about which you are requesting information.

Return Values

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.

Description

Displays information about the macros currently registered with the Calibre layout viewer.

Examples

```
% Macros show  
{CUT REGION} {GRID CHECK} {WRITE HIERARCHY} EXTERNAL {EXECUTE TCL SCRIPT}  
{DEBUG TCL SCRIPT}
```

Related Topics

[Macros create](#)

[Macros delete](#)

[Macros menu](#)

Miscellaneous Batch Commands

There are some general purpose command procedures available for use with Calibre DESIGNrev.

Table 5-19. Miscellaneous Batch Commands Summary

Command	Description
version	Returns the product version string. It can be used in Tcl scripts and at the Tcl prompt to return the product version string.
getCWBOjects	Return handles of each open Calibre DESIGNrev session window.

version

Returns 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.

Return Values

Returns the product version string.

Examples

```
% version  
2014.1_0.15
```

getCWBObjects

Return handles of each open Calibre DESIGNrev session window.

Usage

getCWBObjects

Arguments

None.

Return Values

A Tcl list containing the handles of all currently open windows.

Description

Returns a Tcl list containing handles to all open workbenches. List items are ordered by when each Calibre DESIGNrev window was opened, from oldest to newest.

Appendix A

Example Script

Typically, the Calibre layout viewers are accessed using the Calibre DESIGNrev GUI. Alternatively, 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. It is often useful to develop a script interactively, to see the output of each command as you step through them.

Overview	487
Opening a Data File for Writing	488
Determining the Layers and Topcell	489
Writing a Procedure	490
Opening a Data File for Viewing	491
Review of Executed Commands	492
Creating a Batch Script	494
Creating a Macro	498

Overview

The batch commands for the Calibre layout viewers are Tcl-based commands, providing access to internal application functions. Executing batch commands for the Calibre layout viewers involves working with the Calibre layout viewer extensions to Tcl.

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.

The examples in this appendix show you how to use various Calibre DESIGNrev batch commands, access information in a layout database, and write layout information to a file. Finally, you 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.

Opening a Data File for Writing

Calibre 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.

Procedure

1. Invoke Calibre DESIGNrev in GUI mode, and load a layout of your choosing. For example:

```
calibredrv mylayout.gds
```

2. From the terminal window, press the Enter key to access the Tcl “%” prompt.
3. Enter the following command to open a data file in which to write your results:

```
set fileID [open myoutput.txt w]
```

The Tcl set command tells Tcl to define a new variable “fileID” and to set its value. Tcl recognizes the brackets as indicating the need for command substitution to determine the value to place in the “fileID” variable.

The Tcl open command 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 creates the file if it does not exist and overwrite it if it does exist.

4. Enter the following command to 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.

5. 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 brackets as indicating the need for command substitution and the “\$” as indicating a need for variable substitution.

The object method “\$mylayout file”, is a layout viewer batch command that returns the name of the layout file.

Determining the Layers and Topcell

In this procedure you determine the layers and the topcell of a layout database.

Procedure

1. Enter the following command to write the database layers to the output file:

```
puts $fileID "Layers: [$mylayout layers]"
```

This command writes the string “Layers:” followed by the value or values returned by processing the object method, “\$mylayout layers”. The layers command returns the list of the layers in the database.

2. Enter the following command to write the topcell of the layout to the output file.
Topcells are cells with no references to them.


```
puts $fileID "Topcell: [$mylayout topcell]"
```

This puts command is similar to the previous one, except that it calls a different instance command, “\$mylayout topcell”, which returns the name of the topcell. You also have the option of creating a variable to hold the name of the topcell returned by the instance command. For example:

```
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 left bracket ([), right bracket (]), or dollar sign (\$), you can do so by preceding them with a backslash (\).

In addition, the string “\n” represents a newline character and the string “\t” represents a tab character.

Results

The layers and topcell of the layout database are written to the output file.

Writing a Procedure

You now write a Tcl procedure to print out the children of a given cell and then invoke the procedure. You define a Tcl procedure by specifying what it is called, the arguments it takes, and what it does.

As you do not know the number of levels of cells in the layout database, you need to use either a recursive or an iterative procedure to navigate the cell hierarchy. Recursion is used here, 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 calls 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 in the next cell.

Procedure

1. Type or copy the following Tcl procedure:

```
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  
    }  
}
```

Tcl procedures always take the form:

```
proc procName arguments body
```

Where:

- `proc` is the Tcl command that creates a procedure.
- `procName` is the name of the procedure. In this example the name of the procedure is “`proc_cell_tree`”.
- `arguments` is a list of arguments for this procedure and the specified arguments are always enclosed in braces (`{}`). In this example, the argument list contains four arguments, and within the body of the procedure, these arguments are used as variables:

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

The first two 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.

- *body* contains the commands that will be evaluated when the procedure is invoked and 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 body of the procedure first checks to see if you passed it a cell name. If you did not, the value of *C* is "", which is the default value. In this case, the procedure extracts the cell name using the "\$myleayout 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 brackets. It cycles through the values one at a time, evaluating the commands between the braces ({}).

2. Invoke the previously-created procedure to write the children of the topcell to the file:

```
proc_cell_tree $myleayout $fileID
```

Opening a Data File for Viewing

You now open a data file for viewing.

Procedure

1. Close the data file to which you wrote your results, to save what you have written to the file:

```
close $fileID
```

2. 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.

3. Close the file when you are through viewing the contents:

```
close $fileID
```

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

Review of Executed Commands

Here is 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 mylayout.gds

// Calibre DESIGNrev v2014.1
/
...

% 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_4xaww3wwwwbhkbbsly
--> XNOR2
--> AOI2BB2X1
--> OAI32X4
--> oai32_4xaww3wwwwtjdkunjy

```

```
--> AOI222X1
--> aoi222_1xawwpwwwwl1f2fcy
--> OAI31X1
--> oai31_1xawwewwwwwukxutty
--> AOI21X2
--> aoi21_2xaww9wwwwcvsiymy
...

% close $fileID

% exit

/labs/drvbatch % ls
mylayout.gds
myoutput.txt
```

Creating a Batch Script

Tcl scripts are stand-alone programs written using the Tcl language, plus any language extensions you have available. Calibre layout viewer scripts are Tcl scripts using the Calibre layout viewer Tcl extensions and batch commands.

Procedure

1. You run your layout viewer script in batch mode, by including the pathname for the script in the Calibre DESIGNrev invocation. You 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 is 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.

2. In addition to adding a new Tcl procedure, the commands you 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 discussed, placed into a script format. You can copy and save the following script to a file named *myscript.tcl*:

```
# 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
```

3. Save and close the file.

4. Enter the following command to execute the batch script:

```
% calibredrv myscript.tcl mylayout.gds myoutput.txt
```


Results

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
-->   ao_i32_4xaww3wwwwbhkbsly
--> XNOR2
--> AOI2BB2X1
--> OAI32X4
-->   oai32_4xaww3wwwwtjdkun_jy
--> AOI222X1
-->   ao_i222_1xawwpwwwwl1f2fcy -- Empty cell
--> OAI31X1
-->   oai31_1xawwewwwwwukxutty
--> AOI21X2
-->   ao_i21_2xaww9wwwwcvs_iymy
...

```

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 Calibre layout viewer-supplied macros. These macros provide useful functionality and also serve as examples of the sorts of functionality you can add to the application.

Prerequisites

Assume you have the following simple Tcl procedure you want made into a macro:

Procedure

1. Create the following Tcl script:

```
# Tcl procedure to print out the children of a given cell.
proc myfunction { 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
    }
}
```

2. Add a second proc named “myfunction_plug_in”, which invokes the command procedure from the GUI plug-in:

```
proc myfunction_plug_in { wbHandle window} {
    myfunction
}
```

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.

3. Add the following Tcl call to register the original Tcl procedure as a macro:

```
if {[info exists tk_version]} {
    Macros create "my example" myfunction myfunction_plug_in
}
```

The Macros create Tcl command is only available when running the layout viewer’s GUI, and not when running shell mode. Therefore, the call is wrapped in an if block to avoid errors when the Tcl script is sourced.

4. To add the macro to the layout viewer **Macros** menu, source the macro from your *\$HOME/wbinit.tcl* file, and then invoke the layout viewer:

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


— Symbols —

`[]`, [27](#), [37](#)

`{}`, [28](#)

`|`, [28](#)

`$cwb bindKey` command, [106](#)

`$cwb cget -_layout` command, [113](#)

`$cwb cget -_layoutView` command, [114](#)

`$cwb getMenuPath` command, [117](#)

`$cwb getRulerUnits` command, [118](#)

`$cwb getVisibleLayers` command, [127](#)

`$cwb layoutDeleteClbk` command, [115](#)

`$cwb loadDefaultLayerProperties` command, [129](#)

`$cwb loadLayerProperties` command, [130](#)

`$cwb rulerMeasurements` command, [132](#)

`$cwb runMultiRveOnRveOutput` command, [134](#)

`$cwb runRveOnRveOutput` command, [135](#)

`$cwb selectionCloneToNewLayout` command, [136](#)

`$cwb setDepth` command, [137](#)

`$cwb setLayerVisibility` command, [138](#), [141](#)

`$cwb showWithLayerFilterClbk` command, [144](#)

`$cwb updateDisplay` command, [147](#)

`$cwb viewedLayoutClbk` command, [148](#)

`$cwb zoomAllClbk` command, [150](#)

`$cwb zoomToRectClbk` command, [151](#)

`$L allowupdates` command, [240](#)

`$L ancestors` batch command, [242](#)

`$L AND` batch command, [244](#)

`$L bbox` batch command, [248](#)

`$L cellname` batch command, [250](#)

`$L cells` batch command, [251](#), [308](#)

`$L children` batch command, [252](#)

`$L clips` batch command, [254](#)

`$L clipsout` batch command, [256](#)

`$L connect` batch command, [258](#)

`$L COPY` batch command, [261](#)

`$L COPYCELL GEOM` batch command, [262](#)

`$L create cell` batch command, [263](#)

`$L create clip` batch command, [265](#)

`$L create layer` batch command, [271](#)

`$L create ref` batch command, [274](#)

`$L create text` batch command, [278](#)

`$L create wire` batch command, [281](#)

`$L customLayerDrawOrder` batch command, [284](#)

`$L delete cell` batch command, [285](#)

`$L delete clip` batch command, [286](#)

`$L delete duplicate ref` batch command, [288](#)

`$L delete layer` batch command, [289](#)

`$L delete polygon` batch command, [292](#)

`$L delete polygons` batch command, [294](#)

`$L delete ref` batch command, [295](#)

`$L delete wire` batch command, [301](#)

`$L disconnect` batch command, [304](#)

`$L duplicate cell` batch command, [306](#)

`$L exists cells` batch command, [307](#)

`$L expand cell` batch command, [309](#)

`$L expand ref` batch command, [311](#), [322](#)

`$L file` batch command, [319](#)

`$L flatten cell` batch command, [320](#)

`$L format` batch command, [324](#)

`$L gdsout` batch command, [325](#)

`$L gridsnap` batch command, [330](#)

`$L holdupdates` command, [333](#)

`$L import` batch command, [335](#)

`$L instancedbout` batch command, [337](#)

`$L isLayerEmpty` batch command, [338](#)

`$L ismodified` batch command, [339](#)

`$L isOverlay` batch command, [340](#)

`$L isReferenced` batch command, [341](#)

`$L iterator count` batch command, [342](#), [348](#), [356](#), [357](#)

`$L layerconfigure` batch command, [360](#)

`$L layerFilters` batch command, [361](#)

`$L layernames` batch command, [363](#)

`$L layers` batch command, [365](#)

\$L libname batch command, [368](#)
\$L MASK_LAYER_INFO batch command, [370](#)
\$L maxdepth batch command, [372](#)
\$L modify layer batch command, [373](#)
\$L modify origin batch command, [374](#)
\$L modify text batch command, [376](#)
\$L NOT batch command, [379](#)
\$L oasisout batch command, [381](#)
\$L OR batch command, [390](#)
\$L pushmarkers batch command, [394](#)
\$L query batch command, [398](#)
\$L rdbout command, [407](#), [410](#), [411](#)
\$L readonly command, [408](#)
\$L refcount command, [409](#)
\$L scale batch command, [412](#)
\$L SIZE batch command, [414](#)
\$L SNAP batch command, [415](#)
\$L srefsFromAref command, [416](#)
\$L textout batch command, [420](#)
\$L topcell batch command, [421](#)
\$L transcript batch command, [423](#)
\$L transcript OUTPUT command, [424](#)
\$L transcript RANGE, [425](#)
\$L transcript STATE command, [426](#)
\$L units database command, [427](#)
\$L units microns command, [428](#)
\$L units user command, [430](#)
\$L update, [433](#)
\$L viacell batch command, [434](#)
\$L XOR command, [435](#)
\$O gdsout batch command, [447](#)
\$O oasisout batch command, [450](#)
\$str addToLayout command, [468](#)
\$V cell command, [469](#)
\$V databaseToScreenUnits commands, [471](#)
\$V depth command, [472](#)
\$V exportLayerPalette command, [116](#)
\$V exportView command, [473](#)
\$V micronToScreenUnits command, [474](#)
\$V screenToDatabaseUnits command, [475](#)

— A —

addToLayout command, [468](#)
Ancestors batch command, [242](#)
AND batch command, [244](#)

Append on filemerge, [191](#)
Append on merge, [216](#)
asciiout batch command, [246](#)

— B —

Bbox batch command, [248](#)
bindKey command, [106](#)
Bold words, [27](#)
Brackets in Tcl, [37](#)

— C —

Calibre DESIGNrev as background process, [29](#)
Cell reference
 expand, [311](#), [322](#)
 flattening, [311](#), [322](#)
Cellname batch command, [250](#)
Cells batch command, [251](#)
Children batch command, [252](#)
Clips batch command, [254](#)
Clipsout batch command, [256](#)
Command substitution, [37](#), [40](#)
connect batch command, [258](#)
COPY batch command, [261](#)
COPYCELL GEOM batch command, [262](#)
Courier font, [27](#)
Create Cell batch command, [263](#)
Create Layer batch command, [271](#)
Create layout
 basic command (no input), [160](#)
 from a GDS or OASIS file, [161](#)
 from a LEF/DEF or OpenAccess file, [168](#)
 from an ASCII RDB, [165](#)
 from multiple layouts, [175](#)
 mapped layout, [170](#)
Create Polygon batch command, [272](#)
Create Ref batch command, [274](#)
Create Text batch command, [278](#)
Create Wire batch command, [281](#)
customLayerDrawOrder batch command, [284](#)
cwbWorkBench Object Methods, [104](#)

— D —

Database units, [427](#)
Datatypes, [326](#)
Delete Cell batch command, [285](#)
Delete Clip batch command, [286](#)

Delete Duplicate Ref batch command, [288](#)
Delete Layer batch command, [289](#)
Delete Polygon batch command, [292](#)
Delete Polygons batch command, [294](#)
Delete Ref batch command, [295](#)
Delete Wire batch command, [301](#)
Double pipes, [28](#)
DRC EXTERNAL spacing check, [97](#)
Duplicate cell batch command, [306](#)

— E —

Error handling, [59](#)
Exists cell batch command, [307](#)
Exists layer batch command, [308](#)
Expand cell batch command, [309](#)
Expand ref batch command, [311](#), [322](#)

— F —

File batch command, [319](#)
Flatten cell batch command, [320](#)
Forcerename on file merge, [191](#)
Forcerename on merge, [217](#)
Format batch command, [324](#)

— G —

Gdsout batch command, [325](#), [447](#)
getRulerUnits command, [118](#)
getVisibleLayers command, [127](#)
Gridsnap batch command, [330](#)

— H —

Handle, [37](#)
Handles, [24](#)
Heavy font, [27](#)
High Capacity (HC) mode
 unsupported commands and arguments,
 [100](#)

— I —

Import Layout batch command, [335](#)
Instancedbout batch command, [337](#)
IsLayerEmpty batch command, [338](#)
Ismodified batch command, [339](#)
IsOverlay batch command, [340](#)
IsReferenced batch command, [341](#)
Italic font, [27](#)

Iterator Count batch command, [342](#), [348](#), [356](#),
[357](#)

— L —

layerFilters batch command, [361](#)
Layernames batch command, [363](#), [365](#)
Layers batch command, [365](#)
layout
 object methods, [235](#)
 scale, [412](#)
Layout All batch command, [153](#)
layout assembly, [54](#)
Layout Copy batch command, [154](#)
layout delete batch command, [180](#)
layout droasis, [181](#)
Layout Handles returned, [153](#)
Layout Merge batch command, [216](#)
Layout Overlays ASCII batch command, [220](#)
Layout peek command, [221](#)
layoutDeleteClbk command, [115](#)
LayoutHandle batch command, [448](#)
Layouts batch command, [449](#)
LayoutType batch command, [367](#)
loadDefaultLayerProperties command, [129](#)
Loading shared libraries, [36](#)
loadLayerProperties command, [130](#)

— M —

macro extensions, [97](#)
Macro Plug-in, [92](#)
Macros create command, [477](#)
Macros menu command, [481](#)
MGC_CWB_TMP_DIR environment
 variable, [198](#)
MGC_CWB_TMP_DIR environment variable,
 [202](#)
Minimum keyword, [27](#)
Modify Layer batch command, [373](#)
Modify Origin batch command, [374](#)
Modify Text batch command, [376](#)

— N —

NOT batch command, [379](#)

— O —

Oasisout batch command, [381](#), [450](#)
OR batch command, [390](#)

overlay all batch command, [438](#)
overlay create batch command, [439](#)
overlay delete batch command, [442](#)
overlayCells batch command, [451](#)
overlaylayout batch command, [452](#)
Overwrite on filemerge, [191](#)
Overwrite on merge, [216](#)

— P —

Parent, [40](#)
Parentheses, [28](#)
Peek delete batch command, [454](#)
Peek file batch command, [455](#)
Peek peek batch command, [456](#)
Pipes, [28](#)
Pushmarker batch command, [394](#)

— Q —

Query batch command, [398](#)
Quotation marks, [28](#)

— R —

rdbout, [407](#), [410](#), [411](#)
Recursion, [47](#)
refcount, [409](#)
Rename on filemerge, [191](#)
Rename on merge, [217](#)
runMultiRveOnRveOutput command, [134](#)
runRveOnRveOutput command, [135](#)

— S —

Scale batch command, [412](#)
Screencaptures (\$V exportView), [473](#)
selectionCloneToNewLayout command, [136](#)
setDepth command, [137](#)
setLayerVisibility command, [138](#), [141](#)
Shared libraries, loading, [36](#)
showWithLayerFilterClbk command, [144](#)
SIZE batch command, [414](#)
Slanted words, [27](#)
SNAP batch command, [415](#)
Square parentheses, [27](#)
StringFeature command, [465](#)

— T —

Tcl books, [16](#)
Tcl commands for database objects, [99](#)

Tcl macros, [478](#)
Textout batch command, [420](#)
topcell, [421](#)
Transcript, [424](#)
Transcript STATE, [426](#)

— U —

Underlined words, [27](#)
unique cellnames, [191](#), [205](#)
Units database command, [427](#)
Units microns command, [428](#)
Units user command, [430](#)
updateDisplay command, [147](#)

— V —

Variable substitution, [40](#)
version command, [484](#)
viacell batch command, [434](#)
viewedLayoutClbk command, [148](#)

— X —

XOR command, [435](#)

— Z —

zoomAllClbk command, [150](#)
zoomToRectClbk command, [151](#)

Third-Party Information

Details on open source and third-party software that may be included with this product are available in the *<your_software_installation_location>/legal* directory.

